University of Alberta

Library Release Form

Name of Author: *Cheng Chen*

Title of Thesis: *Interface Design for Web-Based Teleoperation*

Degree: *Master of Science*

Year this Degree Granted: *2001*

University of Alberta

INTERFACE DESIGN FOR WEB-BASED TELEOPERATION

by

Cheng Chen

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Electrical and Computer Engineering

Edmonton, Alberta
Fall 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled ***Interface Design for Web-Based Teleoperation*** submitted by ***Cheng Chen*** in partial fulfillment of the requirements for the degree of ***Master of Science***.

Dedicated to My Parents and Wife

Abstract

This thesis applies leading-edge technologies from the computer graphics, computer vision and intelligent hearing fields, such as foveated wavelet image compression, 3D warping and head related transform function, to web-based teleoperation platform design. Based on these methods, a modularized Internet teleoperation framework is proposed for similar application developments in the future. This thesis proposes a partial solution to the problem of limited Internet communication bandwidth encountered by most web-based teleoperation applications. The application of this solution may make real-time feedback and control stability possible. And sensory substitution technique's application may help to enhance this promise.

Acknowledgements

I would like to give my deep thanks to my supervisor, Dr. Max Q-H Meng for his valuable guidance, unique ideas, and deep insight on the topic of teleoperation technology, which provided me with the inspiration and solid basis for conducting my research. This work would have been impossible without Dr. Meng and Dr. Rao's financial support over the last three years.

Some further gratitude is extended to Dr. Tongwen Chen and Dr. Hong Zhang, who provided sincere academic help whenever I encountered problems. Thank you for your patient discussing about specific topics, such as sonar map construction and image compression. A special thank you also goes to my master program committee: Dr. Xian Liu for his help in revising this thesis.

I would also like to thank my colleagues in Advanced Robotics and Teleoperation (ART) lab, who spent unforgettable days and nights with me in academic study, programming, and platform constructions. They are: Mike Yu Chen, Jianjun Gu, James A. Smith, Peter Xiaoping Liu, Loren Wyard-Scott, Dalong Wang and Yan Liu. Thank you all for your encouragement and suggestion that inspire me to stay on track.

It is hard for me to name everyone, who helped me over the last three years I stayed in ART lab. For those of you that I have forgotten to mention, your help was very much appreciated.

Last but not least, I would like to thank my wife, Yunru Li, for her love and ever-existing care surrounding me, which warms my heart ever and forever. I would also like to thank my parents for their encouragement during my study period in U of Alberta. Thank them for their sacrifice of separating from their son over the past thousands of days, with missing and blessing. Thank them for everything they did for me.

Table of Content

List of Figures

List of Symbols, Nomenclature, or Abbreviations

- Fovea: is a small uniformly sampled disk with high resolution, thereby permitting a detailed analysis of objects lying at the center of the visual fields;
- HRTF: Abbreviation of Head-Related Transfer Function, which is a concise name for a set of effects imposed on the sounds listeners hear by the shape of the outer ears, head, and upper body (Blauert, 1983);
- Thin-Wire: A computation model introduced by Chang E.C et al.. in [10] describing network computer architecture. That is, the model comprises an image server and a client viewer connected by a low-bandwidth line. The server may be assumed to have powerful computational resources if necessary; the client has possibly modest resources;
- Plenoptic function: First introduced by Adelson and Bergen (from the latin root *plenus*, meaning complete or full, and *optic* pertaining to vision) to entitle the pencil of rays visible from any point in space, at any time, and over any range of wavelengths. The plenoptic function describes all of the radiant energy that can be perceived from the point of view of the observer rather than the point of view of the source.

Chapter 1

Introduction

1.1 Historic Review

The Internet connects millions of computers all over the world, giving access to communication, data, pictures, videos and even real images of distant environments. However, only a few examples of real physical interaction with distant places are available at the moment. The goal of web-based teleoperation research is to add a new dimension to the Internet by combining network technology with the capabilities of robots. This way, Internet users could discover and physically interact with places far away from their home or test their control algorithms on a real mobile robot platform.

In September 1994, an ASEA IRb-6 robot was connected to the Internet through a Web Server at the University of Western Australia [51] so that anyone with Web access could control the robot, using a gripper it could manipulate wooden blocks on a table. Just four weeks earlier Ken Goldberg at the University of California, Berkeley had connected a SCARA type robot to the Internet. These were the first physical devices to be connected to the web.

Since then, an enormous effort has been undertaken by hundreds of research labs to push this technology. By a conservative estimate in [50], there are at least five web robots now online. There are many devices other than robots also being connected, like the Interactive Railroad at the University of ULN, Denmark and a demonstration web interface for NASA's Sojourner rover, which will be used in the 2001 rover mission to Mars. Some devices can be made with simple and cheap components such as Lego. A recent addition to the telerobot website is a pan camera made of Lego able to take a full 360 degree panorama of the lab.

1.1.1 Online Web-Robots

Limited by the extend and scope of this thesis, here includes some active robots online as following, which can also be found on web site [28], while the exact number of their kind is varying quickly:

- ARITI- Augmented Reality Interface for Telerobotic applications via Internet: control a 4-DOF robot using a remote computer;
- Australian Telerobot: remotely operate a 6-DOF ASEA manipulator located at the University of Western Australia. Believed to be the first remotely operated industrial robot on the Web;
- Bradford Robotic Telescope: robotic operation of an automated telescope;
- CSC Telerobot: remotely operate a 6-DOF manipulator located at the Carnegie Science Center in Pittsburgh;
- Drinking Maiden Exhibition: telerobotic examination of The Drinking Maiden by Ernst Wenck;
- Eyebot Project: control a manipulator holding a web cam;
- Interactive Model Railroad: remote operation of a real train set in Germany;
- Khep On The Web: a Khepera robot controlled from your web browser, with streaming video;
- Puma-Paint - use a Puma robot to paint with brushes;
- Rhino: let your robotic tour guide show you the Deutsche Museum Bonn (only active for short periods);
- Robotic Garden: robotic plant tending and maintenance over the Internet;
- University of Ballarat Telerobot: internet operation of a robot manipulator;

These telerobot web-sites help to accumulate the basic, but valuable experience of web sites construction and clarify lots of primary concepts in telerobot construction and remote control interface design. With the help of technical supporter and Internet user, they grow up to be robust and easy-use.

1.1.2 Prospective Applications

It is expected that the research of Web-based teleoperation will push forward the progress of many fields, including but not limited to robotics. Fields, which will probably benefit from this method of teleoperation include:

- Entertainment: It is apparent from the reaction of people to Australia's Telerobot, and other Internet devices that many people consider operating them entertaining. A private company, LunaCorp Inc in conjunction with Carnegie Mellon University, plans to launch the first private lunar mission. The project involves landing a pair of teleoperated robotic vehicles on the Moon's surface. Intended customers for the mission include a theme park, television network, commercial sponsors, and scientists.
- Tele-manufacturing: There is a large group at University of California Berkeley with a grant of US$1.3 million in developing an Internet accessible, machining service called CyberCut.
- Training: Providing access to robots and other expensive equipment for training purposes, where purchasing cannot be justified.
- Mining: Teleoperation of underground mining equipment is being practiced at some mines and this technique could be used to operate the equipment from any location.
- Underwater Remotely Operated Vehicles (ROVs): ROVs are subject to time delays, limited bandwidth, and unstructured environments providing an ideal application for supervisory control. These constraints are in many ways similar to those experienced in Internet telerobotics.

A substantial effort is now being devoted to developing Internet devices, however it is still in its infancy. Current devices are largely experimental and none have been used to provide a commercial service.

1.2 Previous Related Research

Besides examples of real robots on the Internet, research has also focused on solving the conceptual and theoretical questions encountered in web-based

teleoperation design. Some efforts were made on control method development and application. Fong et al.. [16] introduced an approach to telerobotic collaborative control, which is different from traditional three layer control partitions (autonomous, supervisory and directive control). An important consequence of this approach is the robot can decide how to use human advice: to follow it when available and relevant; to modify it when inappropriate or unsafe.

Some other relative efforts were made on giving the end-user a higher dimensional sensory perception of remote site environment. This is achieved through insertion of haptic device feedback, which is different from graphics and literal stimulus, such as tactile, pressure and temperature sensor generation. Research in this field is ever increasing. An example of the application of haptic interfaces includes Turner [52] et al..'s work in arm-grounded haptic feedback device in tele-manipulation.

To enrich the traditional feedback approach, as visual and literal one, virtual environment (VE) and telepresence were introduced in teleoperation consequently. Noticeably, Milgram [40] et al. proposed augmented virtual reality (AV) concept, which completely fills the middle ground between complete synthetic and complete real.

All these work greatly encouraged the development of web-based teleoperation research and application. Each field advanced their goals forwards in large or relative moderate laps, though little work has been done in merging such state-of-art works into one practical job for functional testing and performance competition. The last factor in complete web-based teleoperation theory and application is the requirement for a general interface design model construction. This aspect has yet to be properly addressed

1.3 Current Status Analysis

Connecting a teleoperation system to the web increases the accessibility of the device to the end-user. The end users can be geographically distributed and do not need to be in the immediate vicinity of the device to be controlled. A familiar, unique and friendly interface may contribute to the operability and require little

(or no) training. At the same time, a uniform interface may also help to shrink the development cost and leads to zero cost eventually.

However, Web-based teleoperation raises many issues and prohibits use of traditional approaches. Its interface design faces three major problems:

- The network (web) can introduce large time delay for which no upper bound can be guaranteed. Latency introduces phase lag into control system transfer function and reduces real-time control capability;
- The network enables inexperienced and untrained people to guide the robots, increasing risks associated with misuse;
- The web interface has to be easy to understand and to use in order to attract as many people as possible;

By observing the existing real robots on Web, it is easy to conclude that many vehicle teleoperation interfaces are often cumbersome, need significant infrastructure, and require extensive training. Many systems overwhelm with multiple displays of multiple sensors while simultaneously demanding high levels of cognition and motor skill. As a result, only experts can achieve acceptable performance.

In order to make vehicle teleoperation accessible to all users, the operator interfaces must be easy to deploy, easy to understand and easy to use. Specifically, methods must be developed to minimize bandwidth usage, which provides sensor fusion displays, and which optimizes human-computer interface.

1.4 Ideas Behind the Proposed Research

Following thoughts and consideration primarily drive this research:

- ❖ Investigate peer-to-peer human-robot interaction;
- ❖ Add sensory substitution approaches into traditional data display methods;
- ❖ Develop sensor fusion displays suitable for vehicle teleoperation;
- ❖ Create Web-based tools to enable teleoperation by novices without instruction or training.

And the following full-fledged techniques make this work became possible:

- Supervisory level autonomy controls robot from obstacle and indoor navigation;
- Wavelet Transform application in computer graphics and image processing for high data compression ratio;
- Acoustic-psychology research and its application in haptic-interface design and the raising of the soft-haptic concept [39];
- Cross-platform programming language, led by Java and its offspring, like Netscape's JAVA Applet, etc.

1.5 Objectives of this thesis

Sophisticated interfaces for teleoperation have become increasingly important. For some applications, of course, teleoperation is merely a temporary measure until autonomous capabilities improve. In other applications, however, the major purpose of the robot is exploration and human-robot interaction is the main feature driving the application. Thus, it is critical that better interfaces be researched, so that truly integrated and efficient human-robot systems can be built.

The purpose of this thesis draws on work in combining the most mature web-based teleoperation techniques into our system, adapting some advanced concepts and techniques, which originated outside of teleoperation, such as adjustable foveated wavelet image compression, 3D image warping and auditory cued navigation into this field. In parallel to the prosperous of teleoperation research, computer graphics and sensory substitution also made significant breakthroughs. After the introduction of the wavelet concept, new methods germinated from the wavelet root became dominant approaches in signal processing and image compression. Especially, uneven sub-regional image processing method and foveated wavelet methods development most fit for the specific situation, such as teleoperation and thin-wire application. Sensory substitution is a well-known concept, and comprises the basis of haptic interface design. An extension of this concept now pertains to the navigation applications for disability people. It

becomes available to web-based teleoperation only recently by the help of 3D audio generation technology.

The theoretical basis and application in web-based teleoperation will be given in detail in the following paragraph. While, the reasons why these techniques could be imported into web-based teleoperation and become practical, lie in analyzing of current online teleoperation techniques and the characteristics of Internet.

1.6 Thesis Organization

Chapter 1 presents readers about the historic review about Web-based teleoperation and interface design work done before this thesis. Chapter 2 introduces the system architecture and interface modularization of the test system. Chapter 3 briefly presents the origin and theoretical basis of wavelet image compression method and its application in image compression. Foveated extension to conventional wavelet method and its mathematician concept will be provided. Chapter 4 presents the 3D image warping and post-rendering concept, which concentrates on smoothly replay based on limited key-point 3D warping. Chapter 5 works on how to introduce 3D audio into control interface to enrich navigation cues within limited network resource. Chapter 6 focus on sonar data re-rendering, maps building, and sensor fusion, which combine data displaying and merging in the world model proposed by this thesis. Chapter 7 leaves some space for discussing conclusion and future work.

Chapter 2

System Architecture and Interface Modularization

2.1 System Architecture

In this research on teleoperation, a popular mobile robot platform, Pioneer 2 PeopleBot mobile robot (P2PB) has been applied, for development and substantiation purposes designed by *Active*Media Robotics, LLC, using Saphira control software developed by Dr. Konolige et al.. of SRI [29].

Figure 2.1 P2PB mobile robot system ([3])

2.1.1 Motors and position encoders

Pioneer 2's drive system uses high-speed, high-torque, reversible-DC motors. It can move with an approximate maximum speed of 60 cm per second.. Each front drive motor includes a high-resolution optical quadrature shaft encoder that provides 9,850 ticks per wheel revolution (19 ticks per millimeter) for precise position and speed sensing and advanced dead reckoning. The error in distance is about 1 cm per meter; the error in rotation is up to 8 degrees per revolution

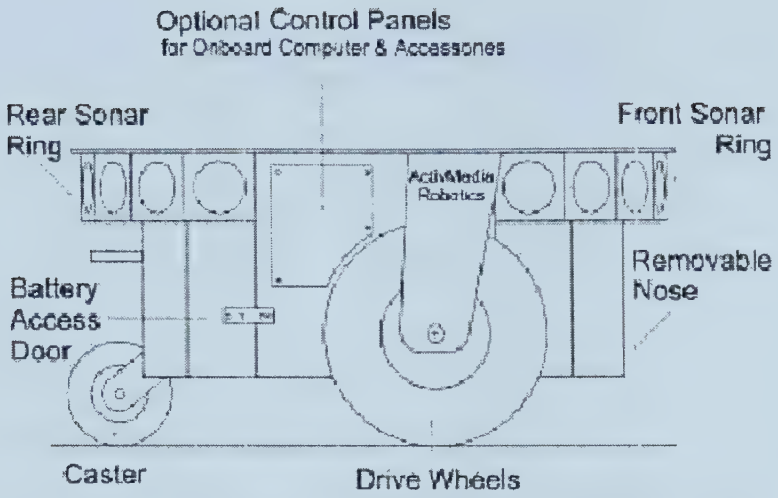

Figure 2.2 Basic Component of P2PB mobile robot.([1])

2.1.2 Sensors

P2PB supports up to two sonar range-finding arrays. One array, affixed under the front of the Deck and atop the Nose, provides forward and side-range sensing. The other, an optional sonar array is attached just beneath the rear Deck and provides rearward, as well as side sensing. Each array contains eight sonars. Total up to sixteen sonars surround the robot. The sonar positions are fixed in both arrays: one on each side, and six facing outward at 20-degree intervals, together providing 360 degrees of nearly seamless sensing.

The sonar-firing rate is 25 Hz (40 milliseconds per sonar per array) and sensitivity ranges from ten cm (six inches) to more than five meters (16 feet). (Objects closer than ten cm are not detected.) Developer may also control the sonar's firing pattern through software; the default is left-to-right in sequence for the forward array (Figure 2.3) and right-to-left on the rear. One sonar from each array "ping"s simultaneously.

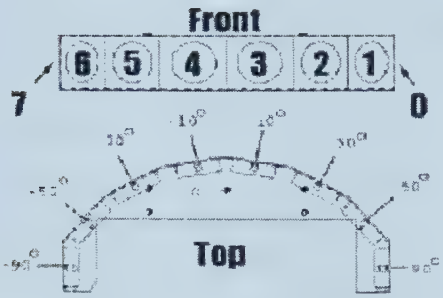

Figure 2.3 P2PB sonar array.([1])

2.1.3 Pan-tilt-zoom Camera

The system mainly consists of a Sony D30/31 pan-tilt-zoom color camera and PTZ system software, which frees us from the detail of camera PTZ control and image grabber. The PTZ camera supplies standard NTSC or PAL video.

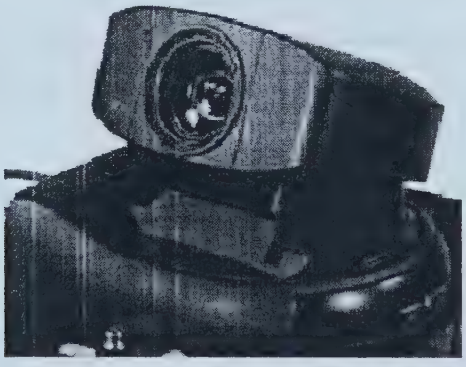

Figure 2.4 PTZ camera system. ([2])

2.1.4 Wireless Ethernet links

A BreezeCom's BreezeNet pro 11 indoor wireless Ethernet card is adapted to execute the communication duty between P2PB and control workstation as shown in Figure 2.5. BreezeNET PRO.11 indoor products adhere to the IEEE 802.11 standard, working seamlessly with other 802.11 Frequency Hopping wireless LAN products. By operating in the 2.4 GHz unlicensed ISM band, it offers data rates of up to 3 Mbps, at distances of up 150m (500ft) indoors, and roaming speeds up to 100km/h (60mph). In real system, a pair of such facilities is applied,

which called as AP-10 (access point to hub slot) and SA-10 (station adaptor fixed with P2PB).

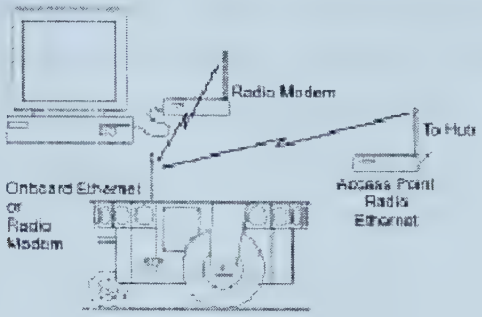

Figure 2.5 Client/Server connection with Wireless Ethernet. ([1])

2.1.5 Console and Electronics

Every Pioneer 2 has a Console that consists of a liquid-crystal display (LCD), MOTORS, and RESET control buttons and indicators, and an RS232-compatible serial port at the 9-pin DSUB connector on the front and top of the Deck.

Figure 2.6 P2PB console and deck layout. ([1])

The P2PB's standard electronics reside on three main boards: The micro-controller is mounted under the Console Deck; a power/motor controller board is mounted to the battery box inside the robot; and a sonar controller (one for each array) is mounted in the base of the sonar array. Please refer to [1] for technical specification and physical parameters.

2.1.6 Sonar sensing

The sonar of the Pioneer exhibits the typical characteristics of sonar sensors; they seldom measure the distance of the nearest object within their main cone. Instead, they often return values that are significantly smaller or larger than the `correct' proximity. Sonar sensors are not particularly noisy; they just do not measure proximity. Instead, they measure the time elapsed between emitting and receiving a focused sound impulse. For smooth objects, chances to receive a sonar echo depend on the angle between the main sonar cone and the reflecting object. Sound waves that hit a wall frontally are very likely to be reflected back in the direction of the sensor, whereas sound that hits a wall in a steep angle is likely to be reflected away in a direction where it cannot be detected. The latter effect is usually referred to as total reflection. As a result, only some of the sonar readings reflect proximity, whereas others do not. In addition, sonar measurements can also be corrupted when the motors draw too much power. These factors introduce unexpected error in sonar map construction.

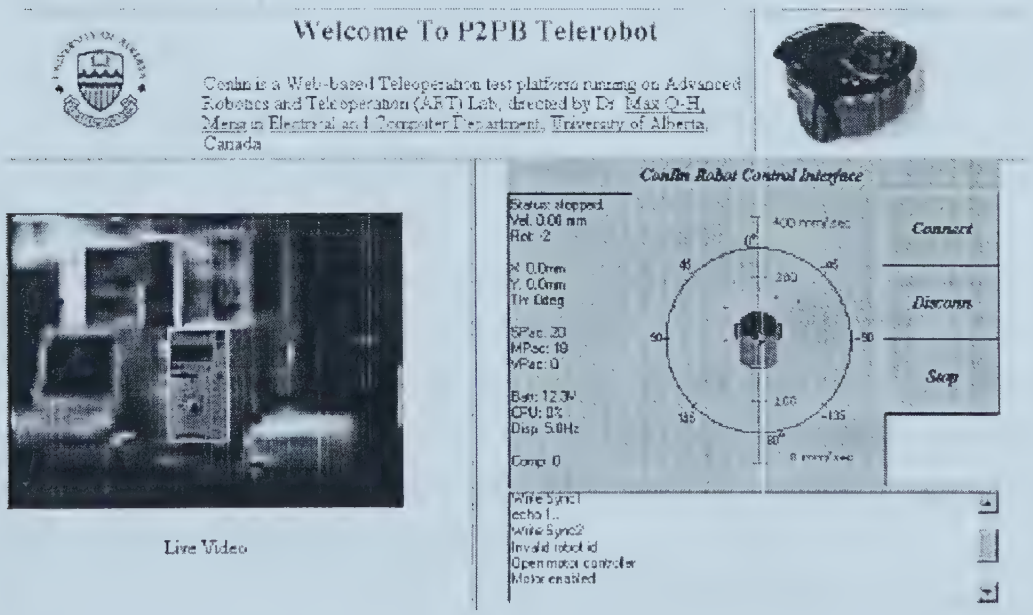

Figure 2.7 P2PB web user interface outlook.

2.2 User Interface Modularization

Being compatible with hardware system, a user interface is designed in orientation to Internet users group, whoever can access Internet with java-enabled Web browser. User interface design concerns the outlook of end data representation methods and user manipulating platform. As well as, operation convenience and feasibility are taken into account with higher priority.

2.2.1 Image Feedback

To improve the responsibility and direct feeling of end-user, real-time image feedback is mandatory even though he/she might be thousands miles away from the site. A live video screen is placed on the center part of the interface, just as shown in Figure 2.8, because of the believing that it is worthy to bring more experience of telepresence than any other ways, even though the transmission of image between worksite and operator via Internet is costly. Foveated wavelet image compression and real-time transfer are implemented here. The end-user can expect a higher resolution display about his/her specified interest area by mouse click and drag. This part information is also transferred as header part of the whole image with first priority. Detail algorithms on foveation and wavelet image compression can be found in Chapter 3.

In fully implemented system, one group of media play buttons will be arranged below the screen, which as shown in Figure 2.8. They are the modifications from traditional teleoperation interface made to replay the histogram by imaged (shot on the trajectory toward target or tour course) based rendering. System will carry the functions of real-time image play and replay on demand in one. The theoretical detail will be explained in Chapter 4.

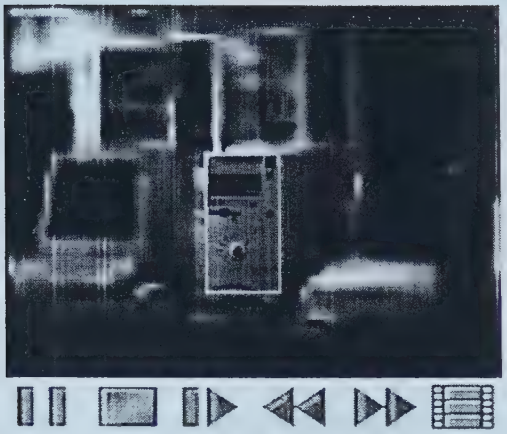

Figure 2.8 Image rendering component on the interface.

2.2.2 World Map

As in computer game, a world map to help localize users current position and conceptual distance between users and their enemy/target is always harmless. If sonar, infrared or other sensor equipments (excluding camera) exist, the upper-right corner screen is always reserved as local or global map. World map may have two sources, where developer (or God) inputs world file pre-installation or generated by sonar data on the fly. For the latter situation, dynamically constructed map also includes the obstacles or other objects surrounding the robot for the purpose of ideal trajectory planning and decision-making. But keeping in mind the possibility of error reading, such as steep wall reflection to ultrasonic wave, obstacles and free space in the map can only be interpreted as a probability index which indicates the risk and safety level of the correspondent location. Fuzzy set theory can be applied to help clarify this blur in Chapter 6.

In this system, sonar map not only plays as sonar data rendering interface and global localization, but also interactive control panel. User may click on the sonar map for next moving direction and how far (in millimeters) it may go scaling from current position as origin.

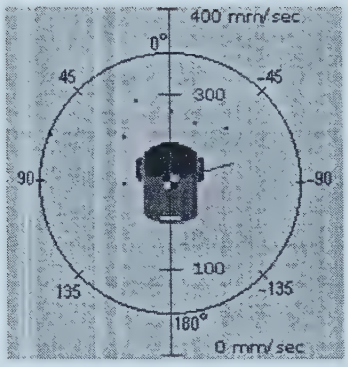

Figure 2.9 A local sonar map captured on the fly.

2.2.3 Instrument Panel

Two additional components will be added in the fully implemented system, which are newly introduced in this control interface. One is auditory cued navigation, shown as a speaker icon, though headphone is preferred in real use to obtain the best quality, which adds auditory aid to operation. It can synthesize 3D audio clip based on spatial position of targets. The target object can be pre-selected or arbitrarily assigned to the nearest object, which may jeopardize the safety of robot.

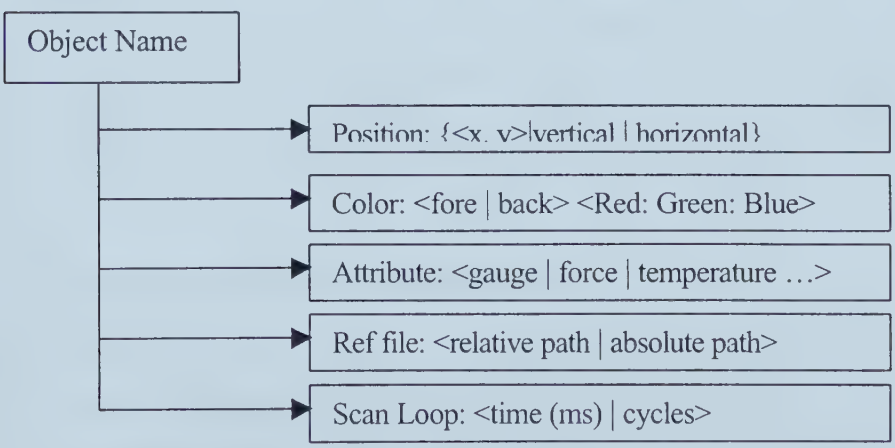

Figure 2.10 Virtual component definition format.

The other is the right-lower equipment window which is extendable depending on the situation of the sever end. Working as a browser plug in, the rendering engine host on client machine by downloading. It provides interface re-rendering based

on server side script file as Web browser does. An appropriate script file may hold such format, which is similar to HTML syntax. To generalize it for multi-platform communication, it will be a plane-text file readable in any circumstance.

- **Object Name:** gives the name of this instruments represents for on server side, say temperature, speed or force. It will show as tip when mouse is over;
- **Position:** is the final position (in X-Y, originated from left-top corner) of the instrument will be placed in the panel layout. A relative expression is also optional, which indicates current component's orientation relative to its neighbors;
- **Color:** can set foreground, background or both with RGB values;
- **Attribute:** specifies the properties it represents, which must be one of the modules in the instruments library. Here, odometer, rotation speed, and thermometer, etc are available right now;
- **Ref File:** indicates the absolute or relative file path, from which the interface engine reads data. The daemon process running on server updates the current reading into the file specified hereby;
- **Scan Loop:** indicates the engine refresh interval on interface. Setup can be in either relative or absolute format. Absolute set time in millisecond, while relative number of dialogue cycles between client and server. This feature helps to decrease the communication load and saves the network bandwidth.

2.2.4 Implementation and Expansion

Running on server end, the daemon process monitors the status of motor, gripper and peripheral instruments. It works as transmitter and executor. To implement the user command and echoing system situation from the uniformed user interface, which may run on diverse systems, for diverse purposes, the process scans sensor reading data and stores them in data files as specified in definition file to rendering system. Easy-use characteristic requires the user's control statements, by graphical pointers on those virtual instruments or input data by

typing, and they will be received and executed by daemon process. The communication channel can be setup in parallel with HTML file download. Three alternative ways are ready for selection; one is TCP/IP socket, which may challenge the safety of remote server, so is too dangerous to be taken into consideration. The other is a little bit meander, while safer, by writing to setting file on server side. This approach is based on JAVA Applet's writing capability assigned from the original server where it is downloaded. A much safer and efficient approach lies on SUN's JAVA Servlet.

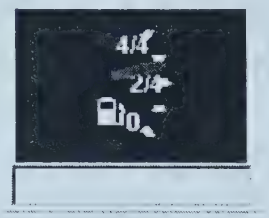

Figure 2.11 Virtual Instrument Panel

Acknowledging the limitation of personal intelligence and capability, and after observing the rising and maturity of some successful software, such as Linux, it seems natural to attribute their success by the means of business and software technology, to their open architecture and free download. These two characteristics assign a software everlasting energy to progress and perfect. The trial version of this research, which is originally given as simple as thermometer for temperature, scale for force or torque, and odometers for speed, etc, will be put on Web for free-download. Each component software package includes data representation and response to user-event function in one, which is similar to ActiveX component/DLL in windows system. Then, some more contributions written in JAVA could be expected from programmers and developers, or even end-users groups. It is hoped that this will help to save those people's time, which intend to construct an economic telerobot Web-site.

Chapter 3

Adjustable Foveated Wavelet Image Feedback

Wavelets are a tool for mathematically decomposing data at different resolutions and then storing the decomposition in a hierarchical manner that lends itself to fast reconstruction. Wavelets provide a better approximation of many functions than more traditional methods of analysis.

3.1 Wavelet Review

The mathematics behind wavelets were developed simultaneously in several fields but were not pulled together into a single theory until later last century. The first mention of wavelets was by A. Haar (1909) as he discussed their usefulness in describing functions with sharp spikes. The overview that follows is based on the wavelet introduction in [25]. Wavelets are currently used to store and analyze fingerprints and compress video for Internet transmission. These typically involve three steps:

1. The application of a wavelet transform to an image to create coefficient matrices;
2. The quantization of these floating point matrices to create integer matrices, and
3. Encoding the quantized matrices to obtain the compressed image.

And the original image can be reconstructed by reversing the steps mentioned above. Some loss in the quality of the image results from the fact that the second step cannot be inverted exactly.

3.2 Mathematics behind Wavelet

Wavelets are mathematical functions used to analyze signals, images or other data representations according to scale. They are used in a manner similar to how sine and cosine are used in Fourier Transforms to approximate functions represented in data. Because Fourier Transforms are non-local, they do not always provide an efficient method for analyzing data, especially when the data contains spikes or discontinuities. The wavelets functions selected to analyze a data function are chosen to best capture both the gross and detailed features.

Wavelet analysis begins with the selection of the wavelet prototype function. This function is contracted along the time-line to produce a high-frequency version to extract detailed information from the data signal. Expanding the prototype wavelet to produce a low-frequency function to analyze gross features of the data produces a second modified wavelet function. The original data function can then be represented by a linear combination of these wavelet functions. These coefficients are the wavelet expansions of the function.

To demonstrate some of the mathematics behind wavelets, take one-dimensional signal or "image" consisting of eight samples (or pixels):

[13 9 8 10 5 9 12 6]

A simple transform will be used for this example:

{Average the pixels together pair wise to get a new lower resolution signal}

This rule captures the gross character of the signal at each resolution level. To capture the details, store

{The difference between each pixel value and its pair wise average}

These two rules form the wavelet transform for the signal. Recursively applying this process on each successive resolution results in this decomposition:

Resolution	Averages	Detail Coefficients
8	[13 9 8 10 5 9 12 6]	N/A
4	[11 9 7 9]	[2 -1 - 2 3]
2	[10 8]	[1 -1]
1	[9]	[1]

Figure 3.1 Recursive Coefficient Compression.

Constructing the wavelet decomposition (or wavelet transform) of the signal is simply taking the value representing the overall signal average and following it with the detailed coefficients:

[9 1 1 -1 2 -1 -2 3]

At this point no data has been lost. The original signal can be reconstructed exactly from its wavelet decomposition. It can also be reconstructed to any intermediate resolution by using its detailed coefficients. The next section will discuss the advantages of storing a wavelet transform instead of the signal or image itself.

Two properties distinguishing wavelet families are orthogonality and normalization. An orthogonal wavelet family has the potential to capture the character of the data in a series of coefficients that can offer greater compression. The coefficients from orthogonal wavelets are not invariant when the input signal is shifted.

The normalization of a wavelet basis provides a metric to be used in determining acceptable error rates or noise in lossy compression.

3.3 Wavelet Image Compression

The magnitudes of coefficients that result from wavelet analysis have a probability distribution resembling a bell curve centered on zero. In other words, a large number of the wavelet coefficients are near zero and, after normalization, are very small in magnitude. Setting a threshold below which to truncate data yields a sparse matrix representing the original data. Adjusting the threshold varies the amount of detail lost in an image when it is reconstructed. A threshold based on the magnitude of the coefficients does not directly relate to the final decompressed data quality.

The simplest technique for determining the threshold is the L^2 norm. The calculations for this norm are greatly simplified if the wavelet basis is orthonormal. Using L^2 norm gives direct feedback into the quality of the final signal or image.

The wavelet coefficients can be quanticized to simplify encoding of the final results. A quantizer is a many-to-one function that maps the input values to a much smaller set of output values. To achieve the best results, a different quantiziation should be applied to the coefficients at each level of resolution.

After the decomposed data has been through any of these lossy compressions, a sparse matrix results that can be further compressed using standard loss-less techniques, such as Huffman coding.

3.4 Foveated Image

Normal images are stored with a uniform resolution, but human vision does not require uniform detail in the field of vision. Less detail is required on the periphery of the vision field than at the focal point. The concept of foveated images and wavelets exploits this feature of natural vision. A foveated image has non-uniform resolution. Areas of interest are viewed at high resolution, while other areas are left at a lower resolution. Thus, substantial savings in both bandwidth requirements and processing time can be obtained by matching the spatial resolution of displayed information to the fall off in spatial resolution of the human visual system.

Variable resolution (foveated) displays are most effective when the direction of gaze is tracked so that the highest resolution region of the display can be kept aligned with the highest resolution region of the eye (the fovea). However, variable resolution displays are also useful in a number of applications where eye tracking is not practical [22], whereas, the location of the high resolution region(s) can be dynamically controlled by the user with a pointing device (e.g., a mouse) or by an algorithm, which is the same situation as this system.

3.4.1 Foveation Theory

There are various practical methods to foveate an image, such as weighted distance, foveation via sample and wavelet approach etc, which are products germinated from the same root. Before diving into the details about individual foveation applications, it is reasonable to repeat some original definition

popularly applied in foveation theory. Different from uniform resolution image, foveated ones characterize themselves by implementing multi-resolution sub-regions in the same image. This effect is achieved by foveation process. According to the discussions in [9], a foveation process is mainly determined by a weight function: $w: R \rightarrow R_{\geq 0}$, where the weight function is required to be positive. A standard weight function is one of the form:

$$w_{std}(t): \alpha \mid t-\gamma \mid +\beta$$

where $\alpha, \beta, and\, \gamma$ are fixed constants, and α is called rate, β the foveal resolution and γ the gaze point. Intuitively, this weight function defines the point $t=\gamma$ as the fovea with maximum resolution β. The resolution decreases linearly, with the distance from the gaze point at the rate α. When $\beta=0$, the resolution at the gaze point is that of the original image. Figure 3.2 gives an example of foveated image comparing with its original format by applying a standard weight function.

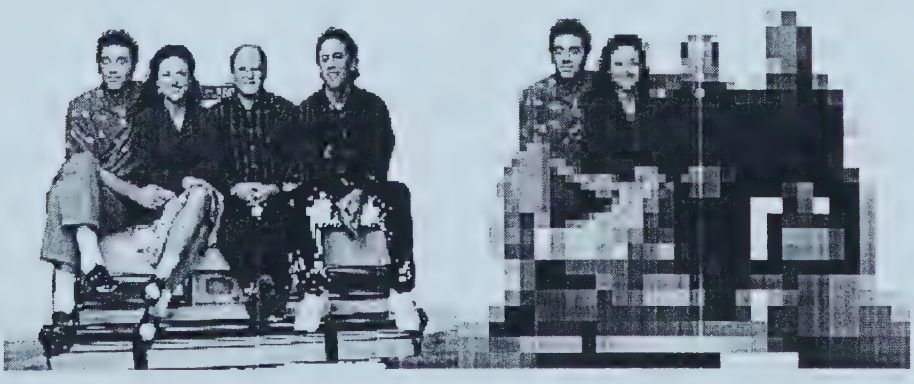

Figure 3.2 (a) Uniform image. (b) Foveation at upper-left corner. ([9])

Starting from weighted function concept, there are several practical foveation approaches proposed in previous works.

Foveation via Sample

Even though the original format of the weighted function by distance from gaze point is given as continuous style, it is very natural to derivate it into discrete format for the sake of computation feasibility and computerization. E. Schwartz et al.. [48] first exploited logmap images for computer vision. In this work, the foveated image is obtained by dividing the visual field into regions called super-pixels (or log-pixels). The value of a super pixel is obtained by averaging all the (original) pixels in the same super cells.

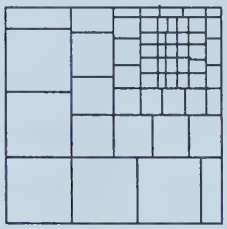

Figure 3.3 A super-pixel arrangement(foveated at north-east)

Foveation via Multi-resolution Pyramid (FMP)

Another valuable approach motivated by foveation theory is filter pyramid, which is studied by Geisler et al.. in [21]. The original image is encoded as a "low-pass pyramid" with 2-6 levels. From each level of the low-pass pyramid an image region is selected. This collection of selected image regions is called a foveated pyramid, and it defines the variable resolution image. The foveated pyramid is the output of the encoder.

Following the other image processing, the foveated pyramid is decoded to obtain a smooth artifact free variable resolution image. Starting with the last level of the pyramid (the lowest resolution), the image is up-sampled, interpolated, and blended with the next highest resolution level. The resulting image is then up-sampled, interpolated, and blended with the next level, and so on, until the highest resolution image is finally blended to complete the foveated image.

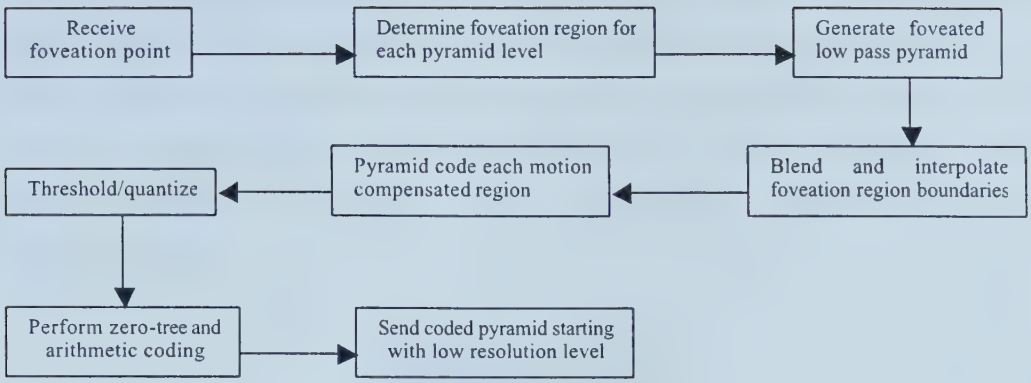

Figure 3.4 Flowchart of Foveated Multi-resolution Pyramid (FMP)

Foveation via Wavelet Approach

Motivated by the multi-resolution frameworks of Mallat, Chang et al. replace standard weight function with wavelet functions in [9], and propose wavelet approach foveation method. Limited by the volume of this thesis, no detail can be extended here for further discussion, but the advantages of this method include fast reconstruction, progressive transmission and easily generalization to higher dimension.

3.4.2 Bandwidth Reduction

Use of foveated image system has demonstrated that significant bandwidth reduction can be achieved, while still maintaining access to high detail at any point of a given image. By the studies of Kortum et al. in [30], using a 20° field-of-view, a half resolution constant of 1° and a foveal Super Pixel size of 1, it will be able to achieve bandwidth reductions of 97.4% (18.8 times reduction) for 256x256 8-bit gray scale images. Increasing the field-of-view or decreasing the half-resolution constant will result in greater bandwidth saving.

3.4.3 Blended Weight

New weight functions can be obtained by combining several given weight functions. Rojer et al. [47] studies the problem of combing several foveated images of a common image into a new foveated image. This is called "blending foveated images". The following formalize definition corresponds to an effective

method for blending images: given weight function w_1 and w_2, their blended weight function w is the point wise minimum of the two functions w(x) = min{ w_1(x), w_2(x)}. Of course, this definition extends to the blending of any finite number of images. The weight function illustrated in is the blending of two standard weights with different gaze points (suitably generalized to two dimension images)

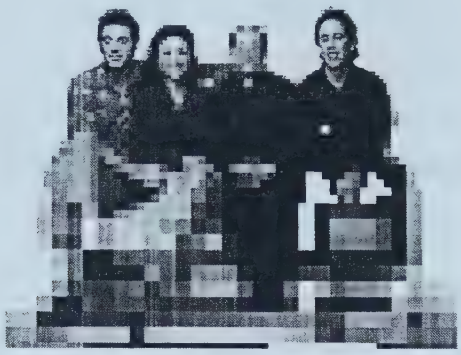

Figure 3.5 Blend image with 2 gaze points. ([9])

3.5 Progressive Transmission

In the case of real-time image retrieval, such as grabbing images on the Internet, as stated in [43], the wavelet compression technique discussed above allows for type of progressive transmission, which is popularly applied by most web-browser for image retrieving. When an image is requested electronically, a wavelet encoded version is brought out of storage, and bits of information about it are sent "over the thin wires" first, starting with the overall average and the larger detail coefficients, and working down to the smallest detail coefficients. As this information is received by the user, it is used to display a reconstruction of image, starting with a very crude approximation of the image that, rapidly updated and refined, looks noticeably better than more wavelet coefficients are used. Eventually (when assuming the user has deemed this picture worth waiting for) all of the detail coefficient will have been transmitted and a perfect copy of image displayed. If the user loses interest or patience along the way, it is easy to halt the process.

Different from the traditional progressive transmission method presented above, progressive transmission in teleoperation hereby combines with foveation theory, which give higher priorities to interest areas. The whole image compressed at once. After compression the compressed image data is ordered in a hierarchical manner. Information describing important image features (sharp edges, large areas) is near the head of the file, while less important information is towards the end of the file. This principle makes it possible to cut off the file in an arbitrary place preserving the most important information of the whole image in the remaining piece. During transmission over a slow network connection a rougher version of the image can be reconstructed before the transmission is completed. As result, this mode will be especially suitable for real-time image feedback working environment over Internet.

Chapter 4

3D Image Warping for Video Replay

4.1 Virtual Reality in Teleoperation

Virtual reality has been introduced into teleoperation field for years. Constrained by network transmission limitation, virtual reality technologies became a practical aid in data display and representation stage, which provide a vivid demo for remote site and help to achieve correct control policymaking. As stated in introduction, virtual environment and real-time image feedback can be regard as both extreme ends of telepresence. Then some middle-stage data representation style also compensates the holes in between, as augmented virtual reality (AV). In AV's definition, geometric models extracted from telerobot are used to reconstruct a semi-live scenario by combining computer graphics model with natural background for simulation purposes, wherever live stream is not available. Regarding AV as a 'forward' simulation based on previous image feedback, it is natural to concatenate passed path together to create a backward correspondent.

As observer enters a new room first time, they may observe the interior world carefully by walking around. Telerobot operator very probably may drive the robot to do the same thing in similar situations. Since the previous experience benefits people's future decision-makings, some high evaluations can also be given to the status of video replay in teleoperation, especially under unknown environment. By observing the past path, operator can be familiar with environment features by rendering stored clips offline. Also, replay concatenates original discrete image frames (because of bandwidth limitation) into a smooth and consequent video stream. Several practical approaches have been constructed for this purpose. A straight forward one is the traditional computer graphics approach to three-dimensional rendering, whereas a scene is described in terms of its geometry and surface properties. Give this representation, the rendering task

can be thought of as a physical simulation problem, which requires huge volume of computation capability and specific hardware support. Recently, there has been an increased interest in alternate rendering paradigm, most of which fall into a category that is called imaged-based rendering.

4.2 Image-Based Rendering

Image-based scene representations provide an alternative to conventional geometrical representations for rendering. Particular interests are given to image-based representations consisting of planar images with per-pixel depth or disparity values. These depth images are compact and can be acquired using computer vision techniques or active depth measuring devices, like laser beam etc, robot compatible equipments.

The frames generated by 3D image grabbing applications exhibit enormous frame-to-frame coherence. This coherence can be exploited to completely avoid conventional rendering of most frames. Every Nth frame is conventionally rendered, and the in-between frames are generated with an image warp that extrapolates from the nearby conventionally rendered frames [35][32]. The original image is referred as a reference image, and the new image as a destination or derived image. This image warp can be used to compensate for latency in the conventional rendering stage. Because the cost of an image warp is largely independent of scene complexity, the technique provides a considerable speedup for complex scenes.

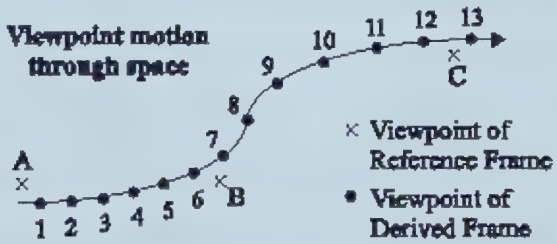

Figure 4.1 Viewpoint motion through space. ([33])

4.2.1 Image Warping

McMillan and Bishop et al. first proposed planar-to-planar, forward image warping algorithm in [35] to compute derived frames from reference frames. They believe the key components of any image warping technique are reference images and mapping functions. For auto-focus stereo applications, estimates of the relative spatial positions are required for the centers of projection of each reference image and an accurate camera model. The required mapping function must somehow capture the geometric nature of the underlying scene. Two possible candidates for this mapping function are the optical flow fields and the stereoscopic disparity fields. These two mappings are nearly equivalent in the case of a static scene, with their only distinction being the size of the baseline. This distinction is nontrivial. It has a dramatic impact on the underlying assumptions that can be made and on the resulting algorithm used to approximate a solution. McMillan's suggestion is the underlying structure of the two vector fields is quite similar.

The image warping function that is used to generate images from new viewpoints has the following general form:

$$x' = F(x, y, \delta)$$
$$y' = G(x, y, \delta)$$
$$I(x, y) \rightarrow I'(x, y)$$

where x and y are pixel coordinates of the reference image, δ is a generalized disparity value, $F()$ and $G()$ are the mapping functions unique to the desired viewing position, and x' and y' are the pixel coordinates of the warped image. The mapping of pixel intensity values is indicated by the third equation.

In order to derive the mapping functions, a description of the projection model assumed must first be given. The planar projection model used in our image warping approach can be described by a point and three vectors, $c, \vec{o}, \vec{u}$ and $\vec{v}$, as shown in Figure 4.2, where the point, c, describes the center of projection, $\vec{o}$ denotes a vector from the center of projection to the coordinate of the origin of the viewing plane, $\vec{u}$ and $\vec{v}$ form a basis set for spanning the view plane. This

formulation is capable of describing any planar projection. It naturally allows for the skewed and off-axis projections required for the display of stereo images on a single display screen.

Only the relative magnitudes of the vectors $\vec{o},\vec{u}$ and $\vec{v}$ are significant. Thus, it is often convenient to scale them such that one or both of $\vec{u}$ and $\vec{v}$ are unit-length.

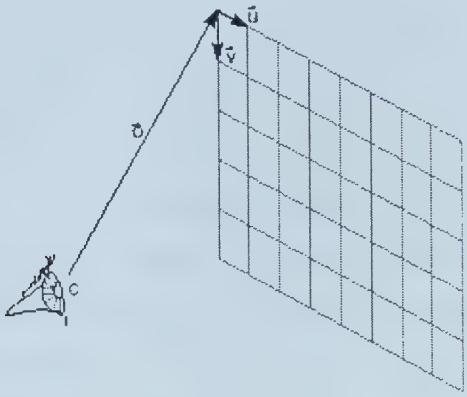

Figure 4.2 Factors determine the planar projection. ([35])

The mapping functions $F()$ and $G()$ now can be specified as the following rational expressions:

$$x' = \frac{ax + by + c\delta + k}{gx + hy + i\delta + m} \qquad y' = \frac{dx + ey + f\delta + l}{gx + hy + i\delta + m}$$

$$\vec{n} = \vec{u} \cdot \vec{\gamma} \qquad \vec{\gamma} = \vec{v} \times \vec{o'} \qquad \vec{s} = \vec{o'} \cdot \vec{\gamma}$$

$$a = \vec{u} \cdot \vec{\gamma} \qquad b = \vec{v} \cdot \vec{\gamma} \qquad c = (\dot{c} - \dot{c'}) \cdot \vec{\gamma} \qquad k = \vec{o} \cdot \vec{\gamma}$$

$$d = \vec{u} \cdot \vec{s} \qquad e = \vec{v} \cdot \vec{s} \qquad f = (\dot{c} - \dot{c'}) \cdot \vec{s} \qquad l = \vec{o} \cdot \vec{s}$$

$$g = \vec{u} \cdot \vec{n} \qquad h = \vec{v} \cdot \vec{n} \qquad i = (\dot{c} - \dot{c'}) \cdot \vec{n} \qquad m = \vec{o} \cdot \vec{n}$$

and the generalized disparity value is given by

$$\delta(x,y) = \frac{\| \vec{o} + x\vec{u} + y\vec{v} \|}{r}$$

Note the similarity between the generalized disparity expression and the classic expression:

$$disarity = \frac{baseline \times focal\ length}{depth}$$

This equation assumes parallel view planes for both reference images and that depth is defined along the optical axis. When this situation cannot be achieved mechanically, image rectification is used to satisfy this assumption. Direct comparisons can be made between the standard disparity measure and the generalized notion. The baseline computation parallels the vector connecting the centers of projection used in the computation of the constants c, f, and i. The role of the focal length is equivalent to the length from the center of projection to a point on the view plane, as seen in the numerator. Depth is replaced by the radial distance, or range value, from the center of projection. Traditional disparity values can easily be converted to the generalized format by dividing out the length of the baseline and multiplying by the tangent of the angle formed by the ray and the optical axis. The primary advantage of this generalized disparity notion is that it allows for arbitrary re-projections.

4.2.2 Visibility Determination

While the image-warping function correctly determines the image coordinate of each pixel in the resulting projection, the possibility exists that it may also introduce many-to-one mappings, called topological folds. Ideally, only the front-most surface would be displayed. Determining the visible surface at each pixel position is one of the fundamental problems of traditional computer vision.

One approach to solving this problem would be to treat the image as a spatial height-field and to use traditional computer graphics techniques to transform and scan convert the result. A standard Z-buffer algorithm could be used to determine the visibility on a pixel-by-pixel basis. There are problems with this approach, though. The transformation process requires the computation of an additional rational expression to determine the z-value for each pixel, and a screen-sized memory array is required to store the z-values. This approach is nearly the same as the transmission of a geometric database described earlier.

McMillan developed an alternative approach to determining the visible surface in [41], which does not require an explicit conversion to a geometric representation. This approach has the following important properties. It determines a unique

evaluation order for computing the image-warp function such that surfaces are drawn in a back-to-front order; thus, it allows a simple painter's style visibility calculation. It maintains the spatial coherence of the image, thus allowing the warp function to be computed incrementally. And, most significantly, the enumeration order can be computed independent of the disparity values for each pixel. This last property is surprising since all of the information concerning the geometry or shape of the underlying data is represented by this disparity information. Since the algorithm is capable of determining the visible surface using only the centers-of-projection of the reference and resulting images, along with the projection parameters of the reference image, this approach allows displayer to use only standard image warping methods without any appeal to the implicit geometric content of the scene.

The space of this thesis does not permit to reiterate every step of the algorithm substantiation, but reciting some important conclusions that pertain to 3D warping. (Please refer to [Appendix A] for discussion)

The algorithm for redisplaying a reference image from an alternate viewpoint, $\ddot{c}$, is described as follows:

1) Compute the projection of the desired center-of-projection onto the reference view plane. This can be computed as follows:

$$x = (\dot{c}' - \dot{c}) \cdot (\dot{v} \times \dot{o}) \qquad y = (\dot{c}' - \dot{c}) \cdot (\dot{o} \times \dot{u}) \qquad w = (\dot{c}' - \dot{c}) \cdot (\dot{u} \times \dot{v})$$

2) Generate a proper enumeration order based on the values of *(x,y,w)*.

3) Warp facets from the original projected grid to the new viewing position.

In order to generate a proper enumeration order based on the value of *(x,y,w)*, several cases must be considered. In the following case analysis, $e' = (x, y, w)$ represents the projected viewpoint or eye position. The enumeration along the i and j directions are considered independently. When the w element of e' is positive, the facets can be enumerated as follows:

- If $x' = x/w$ falls within the range 0 to n-1, then the list of facets must be partitioned into two groups; the first enumerated from $F_{0,j}$ to $F_{x',j}$ and the second ordered from $F_{n-1,j}$ to $F_{x'-1,j}$.

- If $x' = x/w$ is less than zero, then a single ordering from $F_{n-1,j}$ to $F_{0,j}$ is required.
- If $x' = x/w$ is greater than or equal to n, then a single ordering from $F_{0,j}$ to $F_{n-1,j}$ is needed.

Next, the facets are ordered in y according to the following three cases.

- If $y' = y/w$ falls within the range 0 to m-1, then partition the facets into two groups; the first ordered from $F_{i,0}$ to $F_{i,y'}$ and the second ordered from $F_{i,m-1}$ to $F_{i,y'-1}$.
- If $y' = y/w$ is less than zero, then a single ordering from $F_{i,m-1}$ to $F_{i,0}$ is used.
- If $y' = y/w$ is greater than or equal to m, the order is from $F_{i,0}$ to $F_{i,m-1}$.

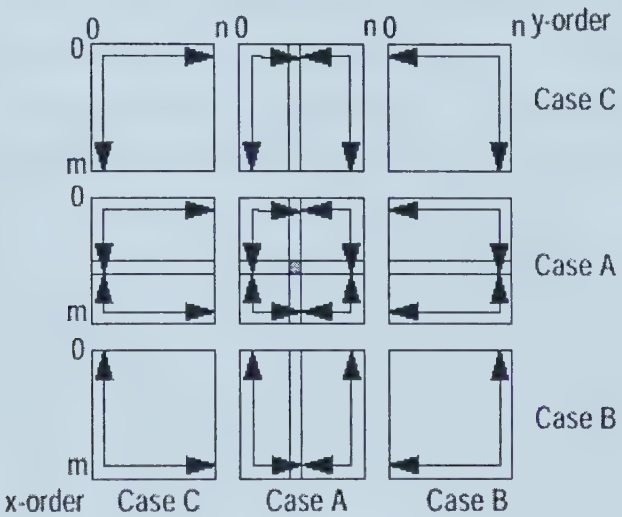

Figure 4.3 Face enumeration order when the value of w is positive. ([35])

Figure 4.3 shows the nine possible enumeration orders after both the x and y orders have been determined. All of the projected grid enumerations are shown relative to the single eye position shown in the center of the figure.

A similar breakdown results when the sign of w is negative. Figure 4.4 shows the nine possible enumerations based on the projected coordinate $(x' y')$.

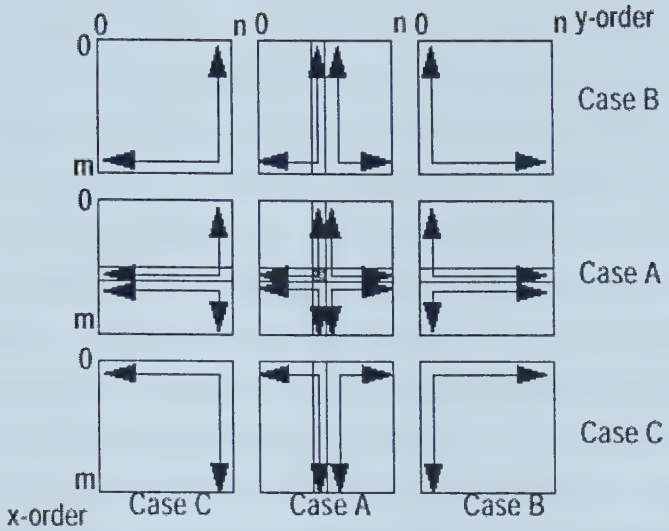

Figure 4.4 Facet enumeration order when the value of w is negative. ([35])

The enumeration order would typically be implemented as nested loops. Thus, the projection of the desired center of projection, *(x' y')*, would first be used to determine whether the surface is to be subdivided into one, two, or four sheets. Next, the starting value, terminal value and increment direction for each of the sheets is assigned. The facets are then painted by looping through the three-space coordinates of the grid points within a given sheet. These grid points are then re-projected to the desired view, V_x. The resulting image will be the correct hidden surface solution for the original projected grid surface when viewed from the new viewpoint.

4.3 Composition and Reconstruction

4.3.1 Reconstruction

The straightforward reconstruction technique of writing a single derived-image pixel for each reference-image sample does not produce images of adequate quality. Mark W. et al. explored two reconstruction techniques that are more sophisticated in [35][32]. The first treats each reference pixel independently, but varies the size of the reconstruction kernel depending on the disparity and normal-vector orientation of the reference pixel. This approach is a form of splat-ting.

This technique's disadvantage is rough edges on under-sampled surfaces and occasional pinholes, which may harm the visual quality of the derived frame.

The second technique treats the reference frame as a mesh. The 3D warp perturbs the vertices of the mesh, possibly causing folds. Reconstruction occurs by rendering the perturbed mesh triangles into the derived frame. The original pixel colors are thus linearly interpolated across the reconstructed mesh element.

A problem with the reconstruction model occurs at silhouette boundaries between foreground and background objects. If the reference-frame mesh is treated as completely continuous, then surfaces are implied at these silhouettes even though the surfaces almost never actually exist. When the reference frame is warped, these implied surfaces manifest themselves as "rubber sheets" stretching from the edge of the foreground object to the background object behind it. The implied surfaces can hide objects behind them, which are visible in a different reference frame, or even elsewhere in the same reference frame. In order to prevent this false occlusion, special treatment is required to these mesh triangles differently from true surfaces. A composition algorithm does this.

4.3.2 Composition

The composition algorithm combines two warped reference frames to generate a single derived frame. At a more detailed level, the algorithm must decide, for each pixel in the derived frame, which of the two warped reference frames will determine that pixel's color. A composition algorithm could blend the contributions from the two reference frames in some cases, but our algorithm always makes a binary decision.

In practice, the 3D warp can produce folds in the warped reference frame (i.e. multiple reference frame pixels are warped to the same location). Therefore, the composition algorithm also arbitrates between multiple potential contributions from the same reference frame, as well as contributions from different reference frames. The most important task of the composition algorithms is to distinguish between surfaces, that actually exist, and the artificial surfaces that are implied by the mesh model at foreground/background silhouettes. Letting an implied surface

incorrectly occlude an actual surface is unwanted, though as would happen in some instances if only relying on depth information for our composition decisions. Some 'depth' definitions and a correspondent algorithm are given here to alleviate this situation.

To disambiguate between the connected and disconnected mesh cases, the notion of connectedness is defined. Each triangle in the mesh is designated as either low-connectedness or high-connectedness, indicating whether or not the triangle is believed to represent part of a single actual surface. Detail discussion about connectedness calculation can be found in [4], which escapes the scope of this thesis. Similarly, A confident notion is given to indicate the ratio between a reference-frame pixel's projected solid angle in the reference frame to its projected solid angle (prior to composition) in the derived frame. The projected solid angle in the derived frame depends on the orientation of the surface to which the pixel belongs.

During the composition process, the derived-image frame-buffers color, Z, connectedness, and confidence information for every derived-frame pixel are ready. This comparison determines whether or not each new pixel should replace the pixel already in the frame-buffer. The decision tree of the algorithm for each pixel is summarized as follows:

1. If both pixels have high connectedness (both belong to valid surface), then compare warped Z values:
    a. If warped Z's are difference the Pixel with closer Z is stored;
    b. If warped Z's are the same (within a tolerance), then Pixel with greatest confidence is stored;
2. If only one pixel has high connectedness (one pixel belongs to a valid surface), then Pixel with high connectedness is stored
3. If neither pixel is high connectedness (neither pixel belongs to a valid surface), then Pixel with greatest confidence is stored. Z's are ignored.

For the sake of performance and user-end computation cost, this algorithm is not applied in real system, but leaving here for discussion purpose only.

4.4 Further extension

Some computer graphics extensions to 3D warping include, predictive tracking, hole filling, anti-alias, etc, which may help to improve the rendering quality to some extent, while with the cost of computer resource consuming. It is rather optional for teleoperation system to exploit them or not, depending on computer capability and end-user's status. Here will just reiterate some hot extension topics, which have been studied by other scientists and their work progress.

4.4.1 Predictive Tracking

Prediction of future viewpoint and view direction can be used in place of, or in conjunction with, image warping to compensate for latency. Predictive tracking has been combined with image shifting by Burbidge in [8]. Predictive tracking alone has been used by many systems. (Refer to [5] for examples and references to other work) Predictive tracking becomes less accurate as the prediction interval increases, and thus becomes less usable (especially by itself) as latencies increase.

4.4.2 Hole filling

When warping any fixed number of source images, some areas of the scene that should be visible in the destination image may not be visible in any source image. These holes in the destination image are usually quite small, but it is crucial that they be filled in a manner that minimizes their perceptual impact.

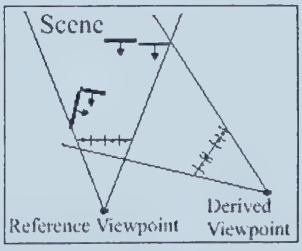

Figure 4.5 Hidden surface in reference frame becomes visible. ([32])

Typically the holes occur where a foreground object has moved with respect to a background surface. A feasible way to fill the hole is thus to extend the

background surface into the hole. Some other methods include multi-resource selection and reference frame average etc.

4.4.3 Anti-alias

Chen and Williams [11] pointed out that reference frames should not be anti-aliased, because the blending of foreground and background colors at silhouette edges is view dependent. Furthermore, the 3D warp requires a single disparity value, and the disparity value of a blended pixel is ambiguous. The same approach to this problem suggested by [12] is implemented: The system generates reference frames at high resolution, warps them, then averages groups of warped samples to produce the derived frame. In other words, super-sampled anti-aliasing is performed where the averaging is deferred until after the warp.

Chapter 5

Auditory Cued Navigation

5.1 Auditory Displays in Actions

Historically, visual and kinesthetic feedbacks have been the primary modes for cuing operator manual control actions in remote manipulator task. Especially, vision displays such an active and stimulating role in our daily lives that often the powerful abilities of our auditory systems go unnoticed. While some facts have been ignored, as human reaction time is faster to a sound stimulus than to a light stimulus [53]. Further, people have a keen ability to detect the direction of sound through binaural discrimination, as well as being able to discriminate sounds due to differences in loudness and pitch [24].

As early as 1936, it was demonstrated that blind folded pilots œuld fly airplanes when two of their instruments indications are presented aurally [13]. In 1983, Tachi, Mann and Rowell proposed auditory display schemes that communicated the course a blind traveler should follow to comply with information a blind mobility aid acquired. Mirchandani et al. also conducted a tracking task with auditory display to evaluate the use of an auditory display for a dual axis compensatory tracking in 1971.

A noticeable recent experiment was conducted by Massimino et al. [36][34] to examine the effect of auditory input on operator performance in a specific telerobotic task. In addition to the visual cues available from stereoscopic cameras and contact force feedback cures to the operator's manual hand controller, they gave the operator an amplified micro-phonic task presentation. The investigation expected to explore whether the amplified sounds in the robot workspace affected human operator performance, within statistical significance of the test object population. And the result supports their premise that adding auditory cue may greatly decrease task operation time and control stability. This example, in

together with the successful experiments cited above, states auditory sensory substitution may play as important as visual and kinematics ones in teleoperation feedback display. Furthermore, their research also support that Auditory display

1. Pose several advantages over visual displays in presenting supplementary information to the operator;
2. Include presenting information irrespective of operator head position and position of gaze;
3. Reduce the need of operator to scan instruments visually which increase the speed with which operator can react to emergency conditions, and requiring extra space on what could already be a crowded control panel.

While, to achieve positive auditory display on teleoperation requires 3D audio synthesis system.

5.2 3D Audio and Generation

5.2.1 Introduction to 3D Audio

A 3D audio system has the ability to position sounds all around a listener. The sounds are actually created by the loudspeakers (or headphones), but the listener's perception is that the sounds come from arbitrary points in space. This is similar to stereo panning in conventional stereo systems: sounds can be panned to locations between the two loudspeakers, creating virtual or "phantom" images of the sound where there is no loudspeaker. However, conventional stereo systems generally cannot position sounds to the sides or rear of the listener, nor above or below the listener. A 3D audio system attempts to do just that.

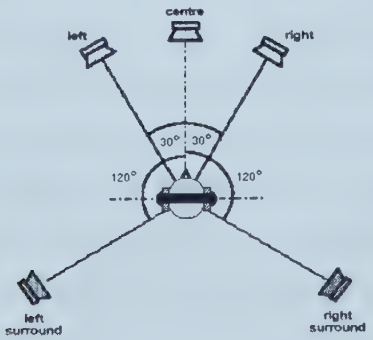

Figure 5.1 3D Audio attempts to create virtual images of sound source.

5.2.2 3D Audio Working Function

To explain 3D audio systems working function, it is useful to start by considering how humans can localize sounds using only two ears. A sound generated in space creates a sound wave that propagates to the ears of the listener. When the sound is to the left of the listener, the sound reaches the left ear before the right ear, and thus the right ear signal is delayed with respect to the left ear signal. In addition, the right ear signal will be attenuated because of "shadowing" by the head. Both ear signals are also subject to a complicated filtering process caused by acoustical interaction with the torso, head, and in particular, the pinna (external ear). The various folds in the pinna modify the frequency content of the signals, reinforcing some frequencies and attenuating others, in a manner that depends on the direction of the incident sound. Thus an ear acts like a complicated tone control that is direction dependent. People unconsciously use the time delay, amplitude difference, and tonal information at each ear to determine the location of the sound. These indicators are called sound localization "cues".

Sound localization by human listeners has been studied extensively in [23]. The transformation of sound from a point in space to the ear canal can be measured accurately; the measurements are called head-related transfer functions (HRTF). The measurements are usually made, by inserting miniature microphones into the ear canals of a human subject or a manikin. A measurement signal is played by a loudspeaker and recorded by the microphones. The recorded signals are then processed by a computer to derive a pair of HRTF (for the left and right ears), corresponding to the sound source location. This process is diagrammed in Figure 5.2. Each HRTF, typically consisting of several hundred numbers, describes the time delay, amplitude, and tonal transformation for the particular sound source location to the left or right ear of the subject. The measurement procedure is repeated for many locations of the sound source relative to the head, resulting in a database of hundreds of HRTF that describe the sound transformation characteristics of a particular head.

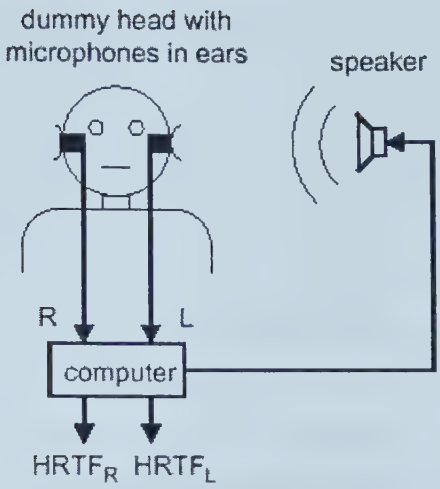

Figure 5.2 Measurement of Head-Related Transfer Function. ([18])

A 3D audio system works by mimicking the process of natural hearing, essentially reproducing the sound localization cues at the ears of the listener. This is most easily done by using a pair of measured HRTF as a specification for a pair of digital audio filters (equalizers). When a sound signal is processed by the digital filters and listened to over headphones, the sound localization cues for each ear are reproduced, and the listener should perceive the sound at the location specified by the HRTF. This process is called binaural synthesis (binaural signals are defined as the signals at the ears of a listener). The binaural synthesis process is diagrammed in Figure 3.2.

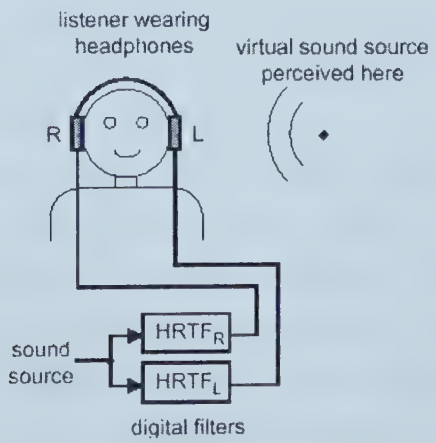

Figure 5.3 Binaural synthesis using HRTF. ([18])

Binaural synthesis works extremely well when the listener's own HRTF are used to synthesize the localization cues [55]. However, measuring HRTF is a complicated procedure, so 3D audio systems typically use a single set of HRTF previously measured from a particular human or manikin subject.

Localization performance generally suffers when a listener listens to directional cues synthesized from HRTF measured from a different head, called non-individualized HRTF. Human heads are all different sizes and shapes, and there is also great variation in the size and shape of individual pinna. This means that every individual has a different set of directional cues. The greatest differences are in the tonal transformations at high frequencies caused by the pinna. It is clear people become accustomed to localizing with their own ears, and thus their localization abilities are diminished when listening through another person's ears. The uniqueness as individuals is the source of the greatest limitation of 3D technology.

The use of non-individualized HRTF results in two particular kinds of localization errors commonly seen with 3D audio systems: front/back confusions and elevation errors[54]. A front/back confusion results when the listener perceives the sound to be in the front when it should be in back, and vice-versa. When 3D audio is reproduced over frontal loudspeaker, back to front confusions tend to be common, which simply means that some listeners may not be able to perceive sounds as being in the rear. In practice, this means that when panning a sound from the front, around to the side, and to the rear, the result will be perceived as a sound panning to the side and then back to the front.

Elevation errors are also common with 3D audio systems. In practice, when a sound is moved from directly to the right to directly overhead, this may be perceived as though the sound is moving from the right to directly in front. This is a typical manifestation of elevation errors, commonly observed when using loudspeakers. Elevation performance is much better when using headphones than when using loudspeakers because the high frequency cues are more faithfully reproduced.

An alternative approach proposed by Gardner et al. is using the HRTF measured from a Knowles Electronic Manikin for Acoustic Research (KEMAR) [17]. The measurements were made in MIT's anechoic chamber. The KEMAR is an anthropomorphic manikin whose dimensions were designed to equal those of a median human. The pinna used was molded from human pinna. In total, 710 measurements were made at different locations around the KEMAR. When synthesizing a location that is not in the measured set, HRTF from four adjacent locations are interpolated.

5.3 Acoustic Environment Modeling

To render the 3D sound sources in real world, system also requires environment acoustic factors for help. Acoustic environment modeling refers to combining 3D spatial location cues with distance, motion, and ambience cues, to create a complete simulation of an acoustic scene. By simulating the acoustical interactions that occur in the natural world, realistic recreations can be achieved stunningly, above and beyond that possible with just 3D positional control [6]. The 3D audio system combines 3D audio with accurate simulations of the following acoustic phenomena: reverberation, distance cues, Doppler motion effect, air absorption, and object occlusion.

5.3.1 Distance Cue

The principal cue for distance is the loudness of the sound. A sound source will be louder when it is closer to the listener than when it is farther away. However, this cue is often ambiguous because the listener doesn't know a priori how loud the source is. Thus, a moderately loud crashing sound could be perceived as a quiet, close crash, or a distant, loud crash.

Another important cue for distance is the relative loudness of reverberation. When sound is produced in a reverberant space, the associated reverberation may often be perceived as a background ambience, separate from the foreground sound [19]. The loudness of the reverberation relative to the loudness of the foreground sound is an important distance cue. The reason for this is due to the acoustics of

reverberant spaces. The foreground sound consists largely of the sound that propagates directly from the sound source to the listener; this so-called direct sound decreases in amplitude as the distance to the listener increases. For every doubling of distance, the amplitude of the direct sound decreases by a factor of one half, or 6 dB. The amplitude of the reverberation, on the hand, does not decrease considerably with increasing distance. The ratio of the direct to reverberant amplitude is greater with nearby objects than it is with distant objects. Thus, distant objects sound more reverberant than close objects.

This relationship is diagrammed in Figure 5.4. The direct sound amplitude drops 6 dB for each doubling of distance (equal to 20 dB drop for a factor of 10 increase in distance). The reverberation amplitude shown below drops at half this slope, or 3 dB per doubling of distance (equal to 10 dB drop for a factor of 10 increase in distance). In most reverberant spaces, the reverberation does not actually drop this fast with increasing distance. However, for the purposes of creating an effective sounding scene, it is often necessary to tweak the parameters to get the desired effect. In particular, when synthesizing virtual acoustic scenes, it can sound unnatural if the reverberation doesn't attenuate sufficiently with increasing distance. It also becomes difficult to localize the sound source if there is too much reverberation.

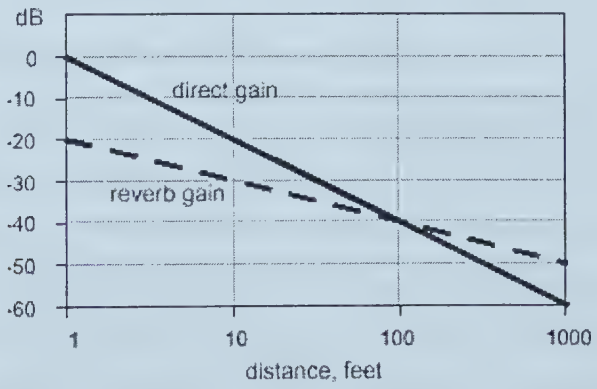

Figure 5.4 Default distance model. ([18])

The relationship between direct and reverberant sound shown in Figure 5.4 is the default distance model used by 3D audio system. For very close distances, the reverberation is 20 dB below the direct sound, equal to a 10% reverb mix. For

increasing distances, the ratio of direct sound to reverberation decreases, and at 100 feet the reverberation is louder than the direct sound. This model is not physically accurate, but produces good sounding results.

5.3.2 Doppler Motion Effect

The Doppler motion effect is commonly heard in nature as a pitch change when a speeding object passes a listener. When the object is approaching the listener, the pitch is higher than the resting pitch of the object. This is because in the time it takes the object to emit one waveform the object has moved closer to the listener, and thus the emitted wavelength is shorter than normal. Similarly, when the object is retreating from the listener, the pitch is lower than the resting pitch, because the emitted wavelengths are longer than normal.

Simulating the Doppler effect is important for generating realistic motion effects. The Doppler motion effect is particularly easy to simulate using a variable delay line. The amount of delay is proportional to the distance between the listener and the sound object. Thus, the delay line effectively simulates the propagation of sound through the air. When the distance changes, so does the length of the delay, and the pitch also changes as it could in nature. Care must be taken that to change the delays smoothly and continuously to avoid distortion and clicks.

5.3.3 Air Absorption

When sound propagates through air, some sound energy is absorbed in the air itself. The amount of energy loss depends on the frequency of the sound and atmospheric conditions. High frequencies are more readily absorbed than low frequencies, so the high frequencies are reduced with increasing distance. For example, at 100 meters distance, 20 degrees Celsius, and 20% humidity, a 4kHz tone will be attenuated by about 7.4 dB [1]. However, the attenuation would be less than 1 dB for distances less than 10 meters. The effect can be simulated by a low pass filter, whose cutoff frequency depends on the distance to the source.

5.3.4 Object Occlusion

When a sound source is behind an occluding object, the direct path sound must diffract (bend) around the occluding object to reach the listener. Low frequencies with wavelengths larger than the size of the occluding object will not be affected much by the occluding object. High frequencies with wavelengths smaller than the size of the occluding object will be shadowed by the object, and will be greatly attenuated. Thus, the effect of an occluding object can be simulated by a low pass filter whose cutoff frequency depends on the size of the occluding object. Simulating object occlusion is important to achieve realism in film/video soundtracks where sound emitting objects are visibly moving behind occluding objects.

5.4 System Design

The 3D audio system is implemented using the signal routing, which is conceptually similar to the routing seen in multi-channel mixing consoles: input signals are individually processed, mixed to a set of shared signal busses, and then the bus signals are processed and output. The input signals represent the individual object sounds that are to be spatially processed to create the scene. If object in the scene has no incident sound attach to it, system may select one basic sound, such as knocking wood, music fork or glass, whichever most similar to the attributes of the object. The input signals are recorded as monophonic.

Each input signal is processed through the Doppler effect, then the air absorption and occlusion effect, and then the 3D spatial effect. The Doppler effect and air absorption effect are controlled by the distance between the sound object and the listener. The occlusion effect is controlled by the position of the sound object, which determines the degree to which the sound object is occluded, and the dimensions of the occluding objects. The 3D spatial effect is controlled by the position of the sound object relative to the listener, which is sampled from remote system via sonar or laser distance sensors. All these data have been sampled before by user specification or remote sonar array. The 3D spatial effect creates stereophonic (two channel) output.

The output from the 3D spatial processor is split into two stereo signals, which are mixed to the "reverb bus" and the "direct bus," each of which is a stereo bus. The amount of sound mixed to each bus depends on the "reverb gain" and "direct gain" mixing gains. These gains are controlled by the distance from the sound to the listener according to the current distance model. Typically, the distance model parameters are set up so that the direct to reverberant ratio increases as the sound object distance decreases.

The reverb bus contains a mix of all sounds that are to be sent to the reverberator. These are processed by the reverberator and the result is mixed with the direct bus. The reverb mix gain determines the overall level of reverberation in the scene. The reverberator is controlled by the scene environment parameters, which include the reverb time, room size, damping, etc. The output of mixer now is ready for rendering in end-user's headphone.

Chapter 6

Map Building and Sensor Fusion

6.1 Map Building Approaches in Mobile Robot

The previous work in robotic map building has revolved around two themes: occupancy or certainty grids, and feature-based methods. Feature-based methods such as demonstrated by Rencken [11] works by locating features in the environment, localizing them, and then using them as known landmarks by which to localize the robot as it searches for the next landmarks. Occupancy grid mapping, as pioneered by Moravec and Elfes ([15],[41]) is a technique that divides the environment into a discrete grid, and assigns each grid location a value related to the probability that the location is occupied by an obstacle. The occupancy grid approach is selected due to its simplicity, robustness and adaptability to dynamic environments.

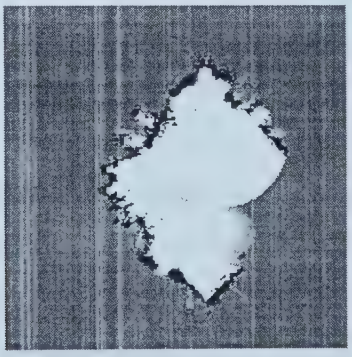

Figure 6.1 Occupancy Grid Map.

In the occupancy grid method, the robot's environment is tessellated into a discrete grid. Each grid location is assigned a value that represents the probability that it is occupied by an obstacle. Initially, all grid values are set to a 50% probability. This represents the "unknown" case. The grid locations that fall within the region of uncertainty about each sensed obstacle point have their values

increased, while locations between the robot and the obstacle have their probabilities decreased.

Several strategies exist for updating grid location values. An intuitive method could be incrementing or decrementing location values with each reading, say 30. Each grid location value could vary from 0 to 255. The increment/decrement step-size is a tunable parameter. A high value allows the map to adapt quickly to new data, but makes it less reliable in the presence of noise, while a low one have the direct opposite cons and pros. Fuzzy logic theory working as the philosophy of map building is prefered. Details about this process will be given in following sections. Here, a sample occupancy grid map is shown in Figure 6.1. In this figure, black represents 100% certain obstacles while white represents clear space.

6.2 Map Building

The map building process is in charge of gathering through the sensors information about the environment at a given robot position and processing it in order to update the available map in accordance. The basic steps are as follows.

- **Perception**: The robot ultrasonic sensors are activated in a proper sequence and a packet of range readings is collected.
- **Processing**: Ultrasonic measures are processed in order to build a local representation of the surrounding scene in terms of empty and occupied space.
- **Fusion**: The local representation is integrated in the global one by filtering out contradictory and insufficient information. In particular, a 256 gray-level bitmap is computed, whereas unexplored areas are regarded as dangerous in map, waiting for further substantiation.

6.2.1 Perception

Ultrasonic range finders measure the distance from obstacles in the environment by a simple conversion of the time of flight of the ultrasonic waves in air. As already mentioned, the mobile robot P2PB is equipped with a ring of 16 ultrasonic range finders. These are constituted by single transducer, which acts both as a transmitter and a receiver. A packet of ultrasonic waves is generated and the

resulting echo is detected. The time delay between transmission and reception is assumed to be proportional to the distance of the sensed obstacle.

A single range reading is affected by three basic sources of uncertainty as follows:

• A sensor has a limited radial resolution. The standard range finder can detect distances from 0.10 to 6 m with 1% accuracy over the entire range.

• The angular position of the object that originated the echo inside the radiation cone is not determined. For example, all the three obstacles of Figure 6.2 will give the same distance reading.

• If the incidence angle is larger than a critical value, the sensor reading is not significant because the beam may reach the receiver after multiple reflections, or even get lost (also see Figure 6.2). The angle depends on the surface characteristics, ranging from 7° to 8° for smooth glass to almost 90° for very rough materials.

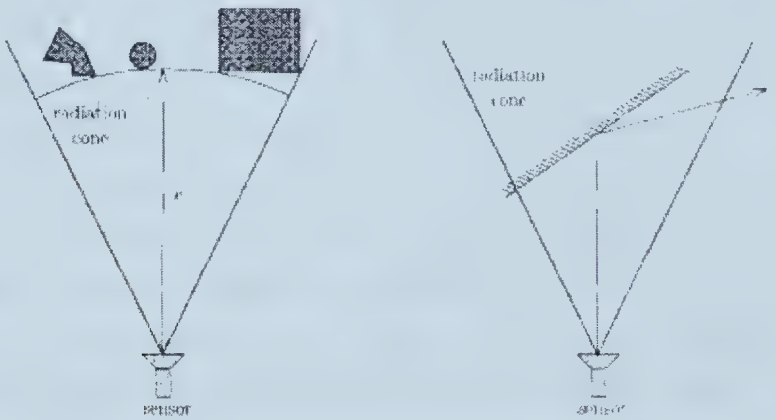

Figure 6.2 Uncertainty introduced by sonar reading. ([44])

During the perception phase, Oriolo suggests that ultrasonic sensors be fired in such a sequence that interference phenomena are minimized, and measures are recorded together with the positions of the corresponding sensors. At each robot position, the ultrasonic ring undergoes two consecutive rotations of 7.5°; as a consequence, 16x3=48 range readings are obtained. Each point of surrounding area falls inside a minimum of three radiation cones. This redundancy of

measures will be exploited in the processing phase in order to achieve a more accurate estimate of the angular position of the detected obstacle.

6.2.2 Processing

The problem of building a map from ultrasonic measures is made difficult by the large amount of uncertainty introduced by the sensing process. This uncertainty consists in a lack of evidence: due to the inherent limitations of ultrasonic sensors, it is not always possible to decide whether a given point of the area of interest is occupied or not by an obstacle. Rather than deciding (i.e., dassifying points of the space as either empty or occupied) in this unfavorable situation, a possible alternative approach is to convey all the available knowledge into an uncertain representation.

Fuzzy logic offers a natural framework in which uncertain information can be handled. Studies on the theory of fuzzy sets started in the early 1970's, with the seminal papers of Zadeh [56]. Define the empty and the occupied space as two status fuzzy sets and over the universal set U (the environment), which is assumed to be a two-dimensional subset of R^2. Their membership functions quantify the degree of belief that the cell is empty or occupied, respectively. This degree of belief should be computed on the basis of the available measures.

In the fuzzy logic context, the two status sets are not complementary—the principle of *tertium non datur (non-middle)* does not hold. Therefore, for a given cell, status sets convey independent information. This situation is particularly convenient in view of the characteristics of the ultrasonic sensing process. In fact, an ultrasonic sensor detects the closest reflecting surface inside its radiation cone, thereby indicating the presence of an empty space up to a certain distance. On the other hand, no information is provided about the state of the area beyond such distance: the available evidence does not suggest emptiness or occupancy. Only by incorporating measures taken at different viewpoints it will be possible to discriminate between the two possibilities.

Processing stage modulate the reading from perception stage according to some presumed axioms as:

1. A single reading provides the information that one or more obstacles are located somewhere along the 25 arc of circumference of radius. Hence, while points located in the proximity of this arc are likely to be occupied, there is evidence that points well inside the circular sector of radius are empty.
2. Since the intensity of the waves decreases to zero at the borders of the radiation cone, the degree of belief of each assertion is assumed to be higher for points close to the beam axis.
3. Finally, since we wish to limit the influence of the range reading to an area close to the sensor location, the degree of belief of the assertions empty and occupied is nonzero only inside a circular sector of radius centered at the sensor position.

By which, the created membership function roughly reflects the previous qualitative description of the sensing uncertainty.

6.2.3 Fusion

During the fusion phase, the local information is aggregated to the global representation of the empty and the occupied space, which is contained in two global fuzzy sets. This process can be implemented as simple as a fuzzy union operation. The final step is to elaborate the information contained in and so as to update local risk map, which is gray-level bitmap conveying information about the risk of collision for each cell of the environment.

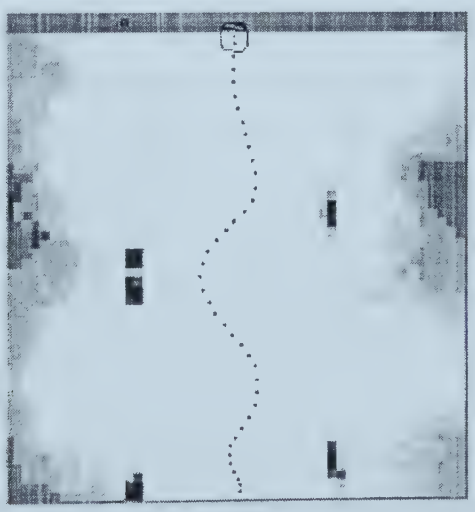

Figure 6.3 A sonar map created with data fusion

6.3 Sensor fusion

Apart from the sonar data fusion mentioned above, to improve teleoperation quality, we also need to make it easier for the operator to understand the remote environment and to make decisions more efficiently. In other words, we need to design the human-robot interface so that it maximizes information transfer while minimizing cognitive loading.

A possible approach is to enhance the quality of information available to the operator. Specifically, we are developing new sensor fusion techniques using multiple sensor data sources to create a unique user interface which efficiently and effectively displays multi-sensor data. In this way, we provide the operator with rich information feedback, facilitating understanding of the remote environment and improve situational awareness.

In [16], Fong et al. used a multi-sensor system with monochrome video, stereovision, ultrasonic sonar, and vehicle odometer. The stereovision system and ultrasonic sonar are co-located on a sensor platform, which is mounted on a vehicle. They chose these sensors based on their complementary characteristics. The stereovision system provides monochrome and range (disparity) images. Ultrasonic sonar provides discrete (time-of-flight) ranges. Though none of the sensors works in all situations, the group as a whole provides complete coverage.

Constrained by software/hardware facilities, we cannot replicate the sensor fusion system, as Fong et al. implemented in their system. While inspired by their thoughts, our sensor fusion display interface will try to incorporate range data with camera images. As shown in Figure 6.4, interface displays sonar ranges as a filled, colored circle (representing the beam cone) image overlay. This may help to correlate the sensor data with real-world obstacles, while operator can observe from image display and help to provide distance cue while navigating between objects surrounding robot.

Some further attempts concentrate on applying color index with sensor reading. As we know, natural colors have different meanings and effects on human

psychology. Many examples, such as traffic lights have been designed to signal our social life, based on these principles. Here, we borrow them to indicate the urgent level of sensor reading to the safety status of robot system. As red means active and dangerous, green defines the opposite extreme, we sort the eight representative colors in between to re-render the sonar data. Each color has big enough interval from its neighbors, for easy recognition by naked eyes.

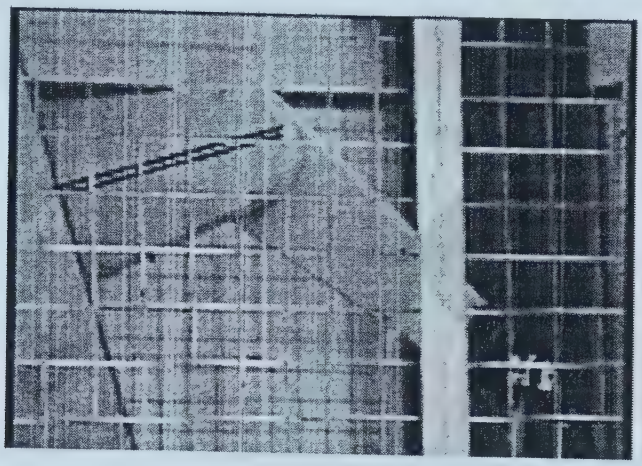

Figure 6.4 Improvement by fusion video and sonar.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Presented in previous is a prototype platform for Web-based teleoperation with indoor mobile robot system. This research enhances the traditional telerobot technology, after the first telerobot system emerged in Australia since 1994, by applying adjustable foveated wavelet image feedback, imaged-based 3D warping for history replay, auditory cued navigation guide extended sensor fusion (sonar and stereo video), and virtual control components into the interface design.

It is found that the operability and performance of manipulating Web-based indoor-mobile robots are increased by introducing these new technologies. Wavelet compression can obtain high ratio (up to 1:20) while still keeps the contour of indoor scene and recognizable. Adjustable foveated image help to decrease this ratio one step forward without lost the detail of interest points. 3D warping video stream replay concatenate separate geometric locations and landmarks together and provide a whole scope of unknown environment. Auditory cued navigation gives fine tune in the process of obstacle avoidance.

During this enhance process, several conceptual presume exist ahead of real works. One is describing Web-based teleoperation as tow equal worlds connected by a thin-wire, where the thin-wire connecting both sides of the real world is Internet. The object is to transfer as much valuable data as possible by software technologies. And the feedback speed is regarded as vital criterion of control safety, as well as the richness of information for proper decision-making. While last, but not the least, is the intention to exorcize those special equipment requirements, which have been attached with web-based teleoperation ever since its birth, from this technology and revert to its original purpose of generosity and plainness.

Some mature computer technologies, such as 3D audio, image-based rendering and foveated wavelet compression, have helped us to achieve that purpose, by providing alternative ways to traditional data representation but extra equipment. From very beginning, the requirements have been restricted for end-user within the scope of monitor, sound card, plus some input devices, as well as mouse and keyboard. It is a believing that these devices will be sufficient for display and manipulate telerobot running and they are adequate for other internet enabled instruments besides robot, which may extends to house holding and daily life fields.

7.2 Future Works

Limited by technical capability and human resources available, this system leaves the 3D warping and acoustic parts in blank. Theoretical experiments substantiates that they are feasible and appropriate to be inserted into such a system, while some technical details still wait for their resolutions. It is also practical problem on how to detect and recognize a remote object on sonar map. This situation may become worse if taking into account the sonar data is fuzzy and inaccurate. And without the 3D sonar data, acoustic cue for object avoidance may become castle in air. A fast vision tracking system may help to alleviate this situation, say by incorporating sonar data with vision object (which is detected by tracking system or human) and 'sticking' sonar echo to that direction. But the calibration and coordination between two sensor systems need further research work.

A compact stereo CCD camera is also mandatory, but in lack of, accessory for this system, which is used to deal with 3D warping. If stereo image included, it is no wonder that some modifications need to be made to fit for the consistency of the system, which will raise new problems, i.e. data compression etc, for solving.

Besides, some further works include constructing global map based on local sonar map histogram for unknown environment, localization and re-calibration in motion, high-density sensor fusion and more data representation component for web devices. The sonar map created in the system will only be used temporarily and follow an as-grabbing-as-dropping process. It is sort of resource wasting if

think of the network cost for transferring these data between two ends of thin-wire. It is expected to reuse these data via concatenating with previous one to construct global map. In this process, some problem may rise automatically as re-localization after slippery, how to reckon the same object in neighbor local map and seamless transit between local maps.

As presented above, the popularity and maturity of Internet technology bring about huge amounts of web-based applications and their derivations. It seems to be a trend for more and more traditional services revised by Internet emerging on web, which hosting in house holding, daily nursing and etc. It is reasonable and valuable to design a standard interface between client platform and background data sampling process, so that provide convenience to prospective programmer and developer. Simultaneously, a uniformed and familiar user interface also benefits end-user performance and improves operation feasibility indirectly. The research in this thesis tries to provide a start point, while far from ending. This work involves standardization processes covering both front-end and back-end application, which means, much endeavor not only emphasis the general look and data manipulating fashion of user-interface, but also data sampling and transmission format.

By and large, Web-based teleoperation development is a comprehensive project. As a big list of research and development fields will be benefited from its prosperous, Web-based teleoperation also requires concerns of different people coving as many as, or even more fields.

Bibliography

[1] *Activ*Media Robotics, LLC, *Pioneer 2 Mobile Robot with Pioneer 2 Operating System Operation Manual*, 2000;

[2] *Activ*Media Robotics, LLC, *PTZ Robotic Camera for Pioneer 2*, 2000;

[3] *Activ*Media Robotics, LLC, *http://www.activmedia.com;*

[4] Airey J., Rohlf J. and Brooks F., *Towards Image Realism with Interactive Update Rates in Complex Building Environments*, ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics, 41-50;

[5] Azuma R. and Bishop G. *A Frequency-Domain Analysis of Head-Motion Prediction*. In Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 95), pp 401–408, Los Angeles, CA, August 1995.

[6] Begault D.R., *3D Sound for Virtual Reality and Multimedia*, Academic Press, Cambridge, MA, 1994;

[7] Bouzeid, G., *Acquisition and Visualization of Ultrasound Data and Sensor Fusion with a 2D Camera*, EPFL Micro-engineering Department, Lausanne, Switzerland, March 1998;

[8] Burbidge D. and Murray P.M.. *Hardware Improvements to the Helmet Mounted Projector on the Visual Display Research Tool (VDRT) at the Naval Training Systems Center*. In Proceedings SPIE, volume 1116, pages 52–60, Orlando, Florida, Mar 1989.

[9] Chang E.C. and Yap C., *A Wavelet Approach to Foveated Images*, Courant Institute of Mathematical Sciences, 1997;

[10] Chang E.C., Yap C.K. and Yen T.J., Real-time Visualization of Large Images over a Thin-wire;

[11] Chen S. E. and Williams L., *View Interpolation for Image Synthesis*. In Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 93), pp 279–288, Anaheim, California, August 1993.

[12] Clark J.H., *Hierarchical Geometric Models for Visible Surface Algorithms*, Communications of the ACM, 19(10):547-554, October, 1976;

[13] deFlorez, L., *True Blind Flight*, Journal of the Aeronautical Science, Vol.3, pp168-170, 1936;

[14] Elfes A., *Occupancy grids: A stochastic spatial-representation for active robot perception*, Proceedings of the Sixth International Conference on uncertainty in AI;

[15] Elfes A., *Using Occupancy Grids For Mobile Robot Perception and Navigation. Computer*, June 1989;

[16] Fong T., Thorpe C., and Baur C., *Advanced Interfaces for Vehicle Teleoperation: Collaborative Control, Sensor Fusion Displays, and Web-based Tools*, Autonomous Robots, July 2001.

[17] Gardner W. and Martin K., *HRTF Measurements of a KEMAR Dummy-Head Microphone*, MIT Media Lab Perception Computing, Technical Report #20, 1994;

[18] Gardner W., *3D audio and Acoustic Environment Modeling*, Technical report, Wave Arts Inc., 1999;

[19] Gardner W.G. and Greisinger, *Reverberation Level Matching Experiments*, Proceedings of the Sabine Centennial Symposium, Cambridge, MA, 1994;

[20] Garlick, D.B. and Winget J, *Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations*, SIGGRAPH '90 Course Notes: Parallel Algorithm and Architectures for 3D Image Generation, 1990;

[21] Geisler W. and Perry J., *Variable-Reduction Displays For Visual Communication and Simulation*, The Society for Information Display, Vol.30, pp 420-423;

[22] Geisler, W.S. and Perry, J.S. *A Real-Time Foveated Multi-Resolution System For Low-Bandwidth Video Communication*, In: B. Rogowitz

and T. Pappas (Eds.), Human Vision and Electronic Imaging, SPIE Proceedings 3299, 294-305. 1998;

[23] Gilkey, R.H. and Anderson T.R., *Binaural and Spatial Hearing in Real and Virtual Environments*, Lawrence Erlbaum Associates, Mahwah, NJ, 1997;

[24] Goldstein, E.B., *Sensation and Perception*, Belmont, CA: Wadsworth, 1989;

[25] Graps A., *An Introduction to Wavelets*, IEEE Computational Science and Engineering, Vol. 2, No. 2, Summer 1995;

[26] Greene, N. and M. Kass, *Approximating Visiblilty with Environment Maps* , Technical Report 41, 1993, Apple Computer Inc.;

[27] Greene, N. *Environment Mapping and Other Applications of World Projections*, IEEE CG&A, Vol. 6, No. 11, November, 1986;

[28] http://ranier.oact.hq.nasa.gov/telerobotics_page/realrobots.html;

[29] Konolige, K. and Myers K. *The SAPHIRA Architecture: A Design For Autonomy*, In Artificial Intelligence Based Mobile Robots: Case Studies of Successful Robot Systems, D. Kortenkamp, R. P. Bonasso, and R. Murphy, eds., MIT Press, 1998;

[30] Kortum, P.T. and Geisler, W.S. *Implementation of a Foveated Image-Coding System For Bandwidth Reduction of Video Images*, In B. Rogowitz and J. Allebach (Eds.) *Human Vision and Electronic Imaging. SPIE Proceedings*, 2657, 350-360, 1996;

[31] Luebke D. and Georges C., *Portals and Mirrors: Simple, Fast Evaluation of potentially Visible Sets*, 1995 Symposium on Interactive 3D Graphics, pp.105-106, April 1995;

[32] Mark W., McMillan L. and Bishop G., *Post-Rendering 3D Warping*, In proceedings of 1997 Symposium on Interactive 3D Graphics, pp 7-16, 1997;

[33] Mark W., and Bishop G., *Efficient Reconstruction Technologies for Post-Rendering 3D Image Warping*, UNC CS Technical Report #TR98-011, 1998;

[34] Massimino M.J., *Sensory Substitution for Force Feedback in Space Teleoperation*, MIT Thesis, 1994;

[35] McMillan L. and Bishop G. *Head-tracked Stereoscopic Display using Image Warping*, In S. Fisher, J. Merritt, and B. Bolas, editors, Proceedings SPIE, Vol. 2409, pages 21–30, San Jose, CA, Feb 1995.

[36] McMillan L. and Bishop G., *Plenoptic Modeling: An Image-based Rendering System*, Computer Graphics Proceedings, Annual Conference Series, pp39-46, Aug. 1995;

[37] McMillan L., *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997. Available as UNC-CH Computer Science TR97-013;

[38] McMillan, L., *A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces*, UNC Technical Report TR95-005, University of North Carolina, 1995.

[39] Meng M., Chen C., Liu P.X. and Rao M., *E-service robot in home healthcare*, 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2000), Takamatsu, Japan, October 2000, vol. 1, pp. 832-837;

[40] Milgram P., and Kishino F., *A Taxonomy of Mixed Reality Virtual Displays*, IEICE Transactions on Information and Systems E77-D, 9 (September 1994), 1321-1329.

[41] Moravec H. and Elfes A., *High-resolution Maps From Wide-Angle Sonar*. In Proc. IEEE Int'l Conf. on Robotics and Automation, St. Louis, Missouri, March 1985;

[42] Moravec H., *Sensor Fusion in Certainty Grids For Mobile Robots, AI Magazine*, pp 61-74 Summer 1988;

[43] Mulcahy C., *Image Compression Using the Haar Wavelet Transform*, Spelman Science and Math Journal, pp22-31;

[44] Oriolo G., Ulivi G. and Vendittelli M., *Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments*, IEEE

Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol. 28, No. 3, June 1998;

[45] Poulakidas, A. Srinivasan, O. Egecioglu, O. Ibarra, and T. Yang, *Experimental Studies on a Compact Storage Scheme for Wavelet-based Multiresolution Subregion Retrieval*, in Proceedings of NASA 1996 Combined Industry, Space and Earth Science Data Compression Workshop, Utah, April 1996;

[46] Regan M. and Pose R., *Priority Rendering with a Virtual Reality Address Recalculation Pipeline,* Computer Graphics Proceedings, Annual Conference Series, pp 155-162, July, 1994;

[47] Rojer A., *Space-variant Computer Vision with Complex-Logarithmic Sensor Geometry*, PHD thesis, Computer Science, Courant Institute, New York University, 1989;

[48] Rojer E. and Shwartz E., *Design Consideration for a Space-Variant Visual Sensor with a Complex-Logarithmic Sensor Geometry*, In 10th ICPR, Vol. 2, pp 278-285, 1990;

[49] Stollnitz E.J., DeRose, T.D. and Salesin D.H., *Wavelets For Computer Graphics: A Primer, Part 1*, IEEE Computer Graphics and Applications, 15(3):76-84, May 1995;

[50] Taylor K. and Dalton B, *Issues in Internet Telerobotics*, Proceedings of International Conference on Field and Service Robotics, 1997;

[51] Taylor K., Trevelyan J., *Australia's Telerobot on the Web*, 26th International Symposium on Industrial Robots, Singapore, October, 1995;

[52] Turner M.L., Gomez D.H., Tremblay M.R. and Cutkosky M.R., *Preliminary Test of an Arm-Grounded Haptic Feedback device in Telemanipulation*, Proceedings of the ASME IMECE Haptics Symposium. Anaheim, CA. Nov. 1998;

[53] Welch R.B. and Warren D.H., *Inter-sensory Interactions*. In handbook of Perception and Human Performance, Chpater 25. Eds. K.R. Boff, L. Kaufman, and J.P. Thomas, New York: John Wiley and Sons, 1986;

[54] Wenzel E.M., Arruda M., Kistler D.J. and Wightman F.L., *Localization Using Non-individualized Head-Related Transfer Functions*, Journal of Acoustic Society, Am. 94(1), pp. 111-123, 1993;

[55] Wightman, F.L. and Kistler D.J., *Headphone Simulation of Free-field Listening. I: Stimulus Synthesis*, Journal of Acoustic. Society, Am. 85(2), pp. 858-867, 1989;

[56] Zadeh L. A., *Outline of a New Approach to the Analysis of Complex Systems and Decision process*, IEEE Trans. System, Man and Cybernetics, Vol. SMC-3, no. 1, pp. 28-44, 1973;

Appendix A Correct Visibility

A.1 Visibility of Projected Surfaces

A perspective-projected surface is distinguished by an origin, a projection manifold, and a scalar range function. This origin, $\dot{e}$, which is often called the *center of projection*, can be considered as the origin of the set of all rays[1] that form the projection. It is convenient to consider the projection manifold, P, as a parameterized surface defined over a space with one less dimension than that of the manifold itself. For the sake of discussion, only three-dimensional manifolds with a parameterization over two-dimensions will be considered. Thus, every point of the manifold, $P(u,v)$, defines a specific ray whose origin is given by, $\dot{e}$, the center of projection[2]. The range function, $R(u,v)$, specifies a length for each ray and is defined over the same parameter space as the projection manifold. Therefore, a perspective-projected surface is itself a parameterized manifold defined over u and v as shown below.

$$S(u,v) = \dot{e} + R(u,v)\frac{P(u,v) - \dot{e}}{|P(u,v) - \dot{e}|}$$

We will consider the problem of re-projecting a perspective-projected surface. This means that, given an initial perspective-projected surface, we would like to generate a consistent new surface with a different origin and/or projection manifold. One difficulty of the re-projection arises from the fact that only a single range value is defined for each ray. However, the re-projection process allows for any number of points from the initial surface to fall along a ray. We resolve this ambiguity by defining the notion of visibility. Visibility dictates that the

---

[1] More precisely is a "*pencil*" of rays.

[2] We disallow the case where the center of projection lies on the projection manifold.

appropriate choice for selecting one from a number of surface points that map to the same ray is to choose the closest one to the desired projection's origin. If we consider the surface to be composed of an opaque material, then visibility corresponds to the natural visual phenomenon of occlusion where far away surfaces along a given ray are hidden by closer surfaces.

One might ask, why should we restrict ourselves to such limited surface representations in the first place? The primary motivation for this choice is that nearly all-natural image formation processes directly record perspective-projected surfaces. This includes photographic images, video images, and the images processed by our visual systems. The goal of re-projecting images is to generate alternate views, which correspond to different view positions.

A.2 Algorithm for determining correct visibility

Next, an algorithm for re-projecting planar perspective-projected surfaces is given in [38]. First, several quantities will be defined in order to describe the algorithm. The original perspective-projected surface is defined by an initial center of projection, e_0, and a planar projection matrix, P_0. It is useful to consider the columns of the projection matrix as the independent vectors $P_0 = [\vec{r}_0 \; \vec{s}_0 \; \vec{o}_0]$. We can then consider P_0 as a ray-generating function over the parametric domain, $\{(u,v) \mid u,v \in [0,1]\}$, where the vector $\vec{o}_0$ is from the center of projection to the parametric origin (i.e. $(u,v)=(0,0)$) of the planar-projection manifold, and, the remaining two vectors, $\vec{r}_0$ and $\vec{s}_0$, span the plane over the parametric domain.

We define the projection of a point, x, by two functions $\Phi_I(e_P, P_P, x)$ and $\rho_I(e_P, x)$, where the parametric mapping function Φ_i determines the parameter space coordinate of the ray passing from the center of projection to the point x, and the range function, ρ_i, gives the distance along this ray to x.

An algorithm for determining the correct visibility of a planar-projected surface when re-projected to an alternate viewpoint, $\dot{e}_1$, is given as follows:

1. Find the projection of the desired viewpoint, $\dot{e}_1$, on the surface to be re-projected $\Phi_0(\dot{e}_0, P_0, \dot{e}_1)$;
2. Partition the manifold into one, two, or four sheets within the parametric domain, according to the isoparametric lines given by $\Phi_0(\dot{e}_0, P_0, \dot{e}_1)$;
3. Enumerate each partition along sequential isoparametric lines toward the positive parametric image of the point, $\dot{e}_1$, while re-projecting onto the new domain Φ_1.

The visibility of the projected surface is determined by the combination of the partitioning and enumeration steps. Essentially these steps establish a visibility-compatible ordering. By this we mean any set of points that map to the same parametric coordinate in the desired view will be ordered such that the last point re-projected will be the one closest to the new center of projection. This ordering is established without any reference to the range values of the surface. Therefore, if the re-projection stage of the third step can also be accomplished without explicit reference to the range function, as in [35] and [38] , the projected scene's visibility can be established without any depth information.

Appendix B Source Code

```
<HTML>
<HEAD>
  <TITLE>Saphira + Java Control Interface</TITLE>
</HEAD>
<BODY>

<div align="center">
 <center>
<table border="2" width="838" height="498">
 <tr>
  <td height="111" width="152">
  </center>
  <p align="center"><img border="0" src=" images/ua.gif" width="87" height="87">
  </td>
  <td colspan="2" height="111" width="529">
 <center>
<h1 align="center" style="line-height: 100%"><font color="#008000" size="5">Welcome To
P2PB Telerobot</font></h1>
  </center>
   <p align="left" style="line-height: 100%">Conlin is a Web-based Teleoperation test platform
   running on Advanced Robotics and Teleoperation (ART) Lab, directed by Dr. <a
href="http://www.ee.ualberta.ca/~max">Max
   Q-H, Meng</a> in <a href="http://www.ee.ualberta. ca">Electrical and
   Computer Department</a>, <a href="http://www.ualberta.ca">University of Alberta</a>,
   Canada.
  </td>
  <td height="111" width="145">
   <p align="center"><img border="0" src="images/redPiowCam.jpg" width="128"
height="110">
  </td>
 </tr>
 <center>
 <tr>
  <td height="293" width="320" colspan="2">
   <p align="center" style="line-height: 100%"><applet code="sock.class" width="320"
height="240" VIEWASTEXT>
                    <param name="host" value="129.128.68.89">
                    <param name="port" value=8104>
                    <param name="interval" value=200>
                    </applet>
   </p>
   <p align="center" style="line-height: 100%">Live Video
  </td>
  <td height="293" width="500" colspan="2">
   <p align="center" style="line-height: 100%"><APPLET CODE=pion WIDTH="400"
HEIGHT="360" VIEWASTEXT>
<PARAM name=host value="129.128.68.89">
<PARAM name=port value=8101>
</APPLET>
  </td>
 </tr>
 </center>
 <tr>
  <td height="76" colspan="2" width="413">
   <p align="left" style="line-height: 100%"><b><font size="2">Usage:</font></b>
   <ol>
```

```
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Connect before execution and
disconnect when logout.</font></li>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Click on sonar map for
movement direction
        and how far to go;</font></li>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Drag and drop on
        live video for interest area
        rendering with finer resolution</font></li>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Right click on right column
image for uniform
        resolution display.</font></li>
    </ol>
   </td>
   <center>
   <td height="76" colspan="2" width="413">
    <p align="left" style="line-height: 100%"><b><font size="2">Attention:</font></b>
    <ol>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">For the sake of p2pb's safety,
please click small step
        on sonar map;</font></li>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Wait till last command executed
before next click;</font></li>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Remember to logout before you
leave.</font></li>
      <li>
        <p align="left" style="line-height: 100%"><font size="2">Any technical question, please
email <a href="mailto:cchen@ee.ualberta.ca">us</a>.</font></li>
    </ol>
   </td>
  </tr>
</table>
  </center>
</div>
<p align="center" style="line-height: 100%"> </p>
</BODY>
</HTML>
```

```
#include <stdio.h>
#include <stdlib.h>

#include <pgm.h>
#include "lwf213.h"

#define  BUFFERSIZE     4096
#define  LICENSE                  "078561-0451136"

typedef struct LWF_tag {
        FILE *file;
        char *buffer;
} LWF;

long LWFAPI flushCoderProc(long flushRefCon, long count) {
        LWF *plwf = (LWF *) flushRefCon;
        if (fwrite(plwf->buffer,1,count,plwf->file)==(size_t)count)
                return LWF_OK;
        else return 1;
}

short LWFAPI seekProc(long flushRefCon, long offset) {
        LWF *plwf = (LWF *) flushRefCon;
        if (plwf)
                return fseek(plwf->file, offset, SEEK_CUR);
        else
                return LWF_SEEK_ERROR;
}

int saveLWF(char *fn, int height, int width, char *pImgData) {
        FILE    *outfile;
        char    cbuffer[BUFFERSIZE];
        LWF     lwf;
        short   LwfRet;

        LwfPictureParams        Pp;             /* picture param object */
        LwfCompressParams       Cp;             /* compress param object */
        LwfMemoryParams                 Mp;             /* memory alloc param object */

        double  rate = 1.0;
        long    comp_size;

        outfile = fopen(fn, "wb");
        if (outfile==NULL) {
                fprintf(stderr, "Sorry, can not create %s, \n", fn);
                return(-1);
        }

        /* we supply a buffer for the file */
        lwf.buffer = cbuffer;
        lwf.file = outfile;

        comp_size = (long)((double)(height*width)/rate);
        /* fill param  for src image param */
        Pp.height = height;
        Pp.width = width;
```

```
        Pp.rowElements = width;
        Pp.baseP = (char *) pImgData;
        Pp.srcColor = CSRC_GRAY;
        Pp.maxValue = (unsigned long)0;
        Pp.dpiH = Pp.dpiV = (long)(72+0.5);

        /* we supply the callback functions to the codec */
        Cp.startLine = 0;
        Cp.endLine = height;
        Cp.stripeInfo = 3;
        Cp.resultP = cbuffer;
        Cp.maxSize = BUFFERSIZE;
        Cp.resultSize = 0;                  /* return from codec */
        Cp.resultCodecQ = 0;                /* same */
        Cp.flushProc = (void *) &flushCoderProc;
        Cp.seekProc = (void *) &seekProc;
        Cp.flushRefCon = (long)(&lwf);
        Cp.maxResult = comp_size;
        Cp.codecQ = 1024;                   /* hi quality */
        Cp.maxLevels = 6;                   /* default value w/ good result */
        Cp.scanMode = 0;                    /* embeded mode */
        Cp.trafoVar = 0;
        Cp.coderVar = 0;
        Cp.enhRectList = NULL;
        Cp.lwfColorSpace = CS_GRAY;
        Cp.lwfPasswordP = NULL;
        Cp.License = LICENSE;
        Cp.reserved = NULL;

        /* initial filling of the buffer from the file */

        LwfRet = LwfGetCompMemoryRequirements(&Pp,&Cp,&Mp);
        if (LwfRet!=LWF_OK) {
                printf("Error: %s \n", LwfGetErrorText(LwfRet));
                return -1;
        }

        printf("prepare mem OK.\n");

        Mp.trafoP = calloc(1, Mp.maxTrafo);
        Mp.coderP = calloc(1, Mp.maxCoder);
        Mp.tempP = calloc(1, Mp.maxTemp);
        Mp.headerP = calloc(1, Mp.maxHeader);
        Mp.koeffP = calloc(1, Mp.maxKoeff);
        LwfRet = LwfCompressLuRaImage(&Pp,&Cp,&Mp);
        if (LwfRet!=LWF_OK) {
                printf("Error: %s \n", LwfGetErrorText(LwfRet));
                return -1;
        }

        fclose(outfile);
        return 0;
}
```

```
import java.util.*;
import Vector3D;

public class Matrix3x3
{
	public float M[];

	public Matrix3x3() {
		M = new float[9];
	}

	public Matrix3x3(Vector3D c1, Vector3D c2, Vector3D c3) {
		M = new float[9];
		M[0] = c1.x; M[1] = c2.x; M[2] = c3.x;
		M[3] = c1.y; M[4] = c2.y; M[5] = c3.y;
		M[6] = c1.z; M[7] = c2.z; M[8] = c3.z;
	}

	public Matrix3x3(float m11, float m12, float m13,
					 float m21, float m22, float m23,
					 float m31, float m32, float m33) {
		M = new float[9];
		M[0] = m11; M[1] = m12; M[2] = m13;
		M[3] = m21; M[4] = m22; M[5] = m23;
		M[6] = m31; M[7] = m32; M[8] = m33;
	}

	public Matrix3x3(Matrix3x3 m) {
		M = new float[9];
		for (int i=0; i<9; i++)
			M[i] = m.M[i];
	}

	public final float element(int row, int col) {
		return M[row*3+col];
	}

	public final void setElement(int row, int col, float v) {
		M[row*3+col] = v;
	}

	public final Matrix3x3 scale(float d) {
		return new Matrix3x3(d*M[0], d*M[1], d*M[2],
							 d*M[3], d*M[4], d*M[5],
							 d*M[6], d*M[7], d*M[8]);
	}

	public final Matrix3x3 plusScaled(Matrix3x3 A, float s) {
		return new Matrix3x3(M[0]+s*A.M[0], M[1]+s*A.M[1],
							 M[2]+s*A.M[2],  M[3]+s*A.M[3],
							 M[4]+s*A.M[4], M[5]+s*A.M[5],
				M[6]+s*A.M[6], M[7]+s*A.M[7], M[8]+s*A.M[8]);
	}

	public final Vector3D times(Vector3D X) {
		return new Vector3D(M[0]*X.x + M[1]*X.y + M[2]*X.z,
```

```
                                                        M[3]*X.x + M[4]*X.y +
M[5]*X.z,
                                                        M[6]*X.z + M[7]*X.y +
M[8]*X.z);
        }

        public final Matrix3x3  transpose() {
                return new Matrix3x3(M[0], M[3], M[6],
                                                        M[1], M[4], M[7],
                                                        M[2], M[5], M[8]);
        }

        public final float determinant() {
                return (M[0]*(M[4]*M[8] - M[5]*M[7]) +
                                M[1]*(M[5]*M[6] - M[3]*M[8]) +
                                M[2]*(M[3]*M[7] - M[4]*M[6]));
        }

        public final Matrix3x3 adjoint() {
                return new Matrix3x3(M[4]*M[8] - M[5]*M[7],
                                                        M[2]*M[7] - M[1]*M[8],
                                                        M[1]*M[5] - M[2]*M[4],
                                                        M[5]*M[6] - M[3]*M[8],
                                                        M[0]*M[8] - M[2]*M[6],
                                                        M[2]*M[3] - M[0]*M[5],
                                                        M[3]*M[7] - M[4]*M[6],
                                                        M[1]*M[6] - M[0]*M[7],
                                                        M[0]*M[4] - M[1]*M[3]);
        }

        public final Matrix3x3 inverse() {
                float m11 = M[4]*M[8] - M[5]*M[7];
                float m12 = M[2]*M[7] - M[1]*M[8];
                float m13 = M[1]*M[5] - M[2]*M[4];
                float d = M[0]*m11 + M[3]*m12 + M[6]*m13;
                if (d==0.0f) return null;

                d = (float)(1.0f/d);
                return new Matrix3x3(d*m11, d*m12, d*m13,
                                                        d*(M[5]*M[6] - M[3]*M[8]),
                                                        d*(M[0]*M[8] - M[2]*M[6]),
                                                        d*(M[2]*M[3] - M[0]*M[5]),
                                                        d*(M[3]*M[7] - M[4]*M[6]),
                                                        d*(M[1]*M[6] - M[0]*M[7]),
                                                        d*(M[0]*M[4] - M[1]*M[3]));
        }

        public final static Matrix3x3 diag(float d) {
                return new Matrix3x3(d, 0, 0, 0, d, 0, 0, 0, d);
        }

        public final static Matrix3x3 identity() {
                return diag(1);
        }

        public final static Matrix3x3 skewSymmetric(Vector3D V) {
```

```
            return new Matrix3x3(0.0f, -V.z, V.y,
                                                    V.z, 0.0f, -V.x,
                                                    -V.y, V.x, 0.0f);
        }

        public final static Matrix3x3 quadSymmetric(Vector3D V) {
            float xy = V.x*V.y;
            float xz = V.x*V.z;
            float yz = V.y*V.z;

            return new Matrix3x3(V.x*V.x, xy, xz,
                                                    xy, V.y*V.y, yz,
                                                    xz, yz, V.z*V.z);
        }

        public final static Matrix3x3 rotation(Vector3D Axis, float angle) {
            float cosx = (float) Math.cos(angle);
            float sinx = (float) Math.sin(angle);

            Axis = Axis.normalize();
            Matrix3x3 R;

            R = diag(cosx);
            R = R.plusScaled(skewSymmetric(Axis), sinx);
            R = R.plusScaled(quadSymmetric(Axis), 1.0f-sinx);

            return R;
        }

        public String toString() {
            return "[["+M[0]+", "+M[1]+", "+M[2]+"],"+
                        " ["+M[3]+", "+M[4]+", "+M[5]+"],"+
                        " ["+M[6]+", "+M[7]+", "+M[8]+"]]";
        }
}
```

```
public class param{
        public static final float AngleConvFactor=(float) 0.001534;      //radians per encoder
count diff (2PI/1024)
        public static final float DistConvFactor=(float)0.826;           //5in*PI / 7875 counts
(mm/count)
        public static final float VelConvFactor=(float)1;
        //mm/sec / count  (DistConvFactor * 50)
        public static final float RobotRadius=(float)250.0;                    //radius in mm
        public static final float RobotDiagonal=(float)120.0;                  //half-height to
diagonal of octagon
        public static final int         Holonomic=1;
        //turns in own radius
        public static final float MaxRVelocity=(float)100.0;                   //degrees per
second
        public static final float MaxVelocity=(float)400.0;                    //mm per second

        public static final float RangeConvFactor=(float)0.268;                //sonar range mm
per 800 ns tick
        public static final int         ImageCenterX = 100;
        //center point X of image
        public static final int         ImageCenterY = 120;
        //center point Y of image
        public static final int         ImageRangeMax = 100;                           //image
max range b/w cntr & bound
        public static final int         ImageRangeMin = 0;
        //imgae min range b/w cntr & bound
        public static final int         ObjSize = 1;
        //image object size
        public static final int         SonarUnitTh[] = {90,50,30,10,-10,-30,-50,-90,
                                                                                -90,-
130,-150,-170,170,150,130,90};
}
```

```
/*
 * This program reads from stdin an image file coded in pgm (portable
 * graymap) in RAWBITS into an array, allocates storage for a second
 * pgm file, copies the first image to the second image, and then writes
 * it to the output (stdout), all using standard pgm library functions.
 */

#define THRESH 200

#include <stdio.h>
#include <pgm.h>

void main( argc, argv )
int  argc; char *argv[];
{
 int grad;
 int rows,cols;
 int i,j;

 gray maxval;                /* type gray is unsigned short */
 gray **graymap;               /* graymap is the 2D array
                          holding the pixels */
 gray **edgemap;               /* edge map */

 /*
  * Read into array from stdin
  */

 graymap = pgm_readpgm(stdin, &cols, &rows, &maxval);

 /* Allocate storage for another image file */

 edgemap = pgm_allocarray(cols, rows);

 /* Copy the first to the second */

 for(i = 1; i < rows-1; i++)
   for(j = 1; j < cols-1; j++)
     edgemap[i][j] = graymap[i][j];

 /*
  * Write the second file to stdout
  */

 pgm_writepgm(stdout, edgemap, cols, rows, 255, 0);
}
```

```
import java.net.*;
import java.io.*;
import java.awt.*;
import java.applet.Applet;
import java.util.*;

public class pion extends Applet implements Runnable{
        int     port;           /* port number */
        String  host;   /* host IP addr */

        Thread  t_pulse;

        InetAddress     ia_server;      /* internet server */
        Socket  sock;                   /* socket */
        /*Input and output stream via socket */

        static      DataInputStream         input;
        static      DataOutputStream output;


        TextArea            ta_Status;
        TextArea            ta_Cmd;
        Button              bt_Conn;
        Button              bt_Disc;
        Button              bt_Stop;
        TextField           tf_Cmd;
        ImagePanel  p_image;
        pPTZ                ptz;

        static boolean      b_connected = false;

        int     SonarUnitTh[] = new int[16];
        float   sonar_read[] = new float[16];
        float   Xpos=0, Ypos=0;
        int             Thpos=0;
        int             Compass=0;
        int             Control=0;
        int             degSet=-1;
        float   tgtX=0, tgtY=0;

        int             status=0;
        float   f_Dist = 5000;
        int             motion=0;       //stop

 public void init() {

  host = getParameter("host");
  if (host == null) host = "conlin.ee.ualberta.ca";

  String s_Port = getParameter("port");
  if (s_Port == null) port = 8081;
  else port = Integer.parseInt(s_Port);

        setLayout(new BorderLayout());

  //System.out.print("\nHost and ports: ");
```

```
        for (int i=0; i<16; i++) sonar_read[i] = 0;

        ta_Status = new TextArea("Status:\t no conn \n" +
                                                        "Vel:\t 0 \n" +
                                                        "Rot\t 0 \n\n" +
                                                        "X:\t 000000 \n" +
                                                        "Y:\t 000000 \n" +
                                                        "Th:\t 000 \n \n" +
                                                        "MPac:\t 0 \n" +
                                                        "SPac:\t 0 \n" +
                                                        "VPack\t 0 \n\n" +
                                                        "Bat:\t25.0v \n" +
                                                        "CPU:\t 0% \n" +
                                                        "Disp:\t5.0Hz \n\n" +
                                                        "Comp:\t 000", 16, 12,
TextArea.SCROLLBARS_NONE);
        ta_Status.setEditable(false);
        add(ta_Status, BorderLayout.WEST);

        p_image = new ImagePanel(ge tCodeBase(),"images/pion.gif");
        add(p_image,BorderLayout.CENTER);

        Panel p = new Panel();
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints con = new GridBagConstraints();
        con.fill = GridBagConstraints.VERTICAL;

        p.setLayout(grid);
        con.gridx = con.gridy = 0;
        p.add(bt_Conn = new Button("Connect"), con);
        con.gridx = 0;
        con.gridy = 1;
        p.add(bt_Disc = new Button("Disconn"), con);
        con.gridx = 0;
        con.gridy = 2;
        p.add(bt_Stop = new Button("Stop"),con);
//      bt_Disc.setEnabled(false);
//      p.add(tf_Cmd = new TextField(10));
//      tf_Cmd.setEnabled(true);
        con.gridx = 0;
        con.gridy = 3;
        con.ipady = 20;
        ptz = new pPTZ();
        p.add(ptz, con);
        add(p, BorderLayout.EAST);

        ta_Cmd = new TextArea(">>", 4, 12, TextArea.SCROLLBARS_VERTICAL_ONLY);
        add(ta_Cmd, BorderLayout.SOUTH);
        Font f = new Font("Serif", 3, 14);
        setFont(f);
  add(new Label("Conlin Robot Control Interface",Label.CENTER),BorderLayout.NORTH);

        try {
                ia_server = InetAddress.getByName(host);
                sock = new Socket(ia_server, port);
```

```
                dispMsg("connected to " + host + "...\n");
                input = new DataInputStream(sock.getInputStream());
                output = new DataOutputStream(sock.getOutputStream());
                ptz.PTZInit();
            }
            catch (Exception e) {
              dispMsg("Connection Failed !!!\n");
              e.printStackTrace();
            }
          }

          private int calc_chksum(byte[] ptr) {
                /* ptr is array of bytes, first is data count */
                int n;
                int c = 0;
                int offset=2;

                n = ptr[offset];
                offset ++;
                n -= 2; /* don't use chksum word */
                while (n > 1) {
                        c += (int)(ptr[offset]<<8) | (int)(0xff&ptr[offset+1]);
                        c = c & 0xffff;
                        n -= 2;
                        offset += 2;
                }
                if (n > 0) c = c ^ (int)ptr[offset];
                return(c);
              }

          private int parseCmd(byte[] indata) {
                int offset = 0;
                int len = 0;
                String ret;
                String echo="Name: ";

                if (indata[0] != (byte)0xfa || indata[1] != (byte)0xfb)  return -1;

                offset += 2;
                len = (int) indata[offset++];
                if (indata.length <= len+2) return -1;

                int chk_sum = ((byte)indata[len+1]<<8) | ((byte)indata[len+2]);
          //    if (chk_sum != calc_chksum(indata)) return -1;

                ret = new String(indata, 3, len-2);

                if (!b_connected) {          /* sync info */
                        switch (indata[offset]) {
                        case      0:
                          dispMsg("echo 0...\n");
                          break;
                        case      1:
                          dispMsg("echo 1...\n");
                          break;
```

```
                    case 2:
                     try{
                              StringTokenizer st = new StringTokenizer(ret," \0");
                              dispMsg("NAME: " + st.nextToken() +" CLASS: " + st.nextToken() +
                                       " SUBCLASS: " + st.nextToken() +" \n");
                     } catch(Exception e) {
                               dispMsg("Invalid robot id\n");
                     }
                     b_connected = true;
                     break;
                    }
          }
          else {
                    dispMsg(parseSIP(new DataInputStream(new
ByteArrayInputStream(indata))));
          }
          return (int)(indata[offset]);
 }

 private byte[] getSync(int iSyncId) {
          byte data[] = new byte[6];

          data[0] = (byte)0xfa;        /* header part */
          data[1] = (byte)0xfb;

          data[2] = (byte) 3;
          data[3] = (byte) iSyncId;

          int chksum = calc_chksum(data);
          data[4] = (byte)(chksum>>8);
          data[5] = (byte)(chksum);

          return data;
 }

 private byte[] getCmd(int iCmd, int iType, byte [] argdata) {
          byte data[]=null;
          int offset;

          if (argdata==null) {          /* pulse|open|close arg*/
                    argdata = new byte[2];
                    switch(iCmd) {
                    case 0:
                              argdata[0] = (byte)0x07;
                              argdata[1] = (byte)0x0f;
                              break;
                    case 1:
                    case 2:
                              argdata[0] = (byte)0x01;
                              argdata[1] = (byte)0x00;
                              break;
                    }
          }

          data = new byte[argdata.length+7];
          data[0] = (byte) 0xfa;                    /* header part */
```

```
        data[1] = (byte) 0xfb;
        data[2] = (byte) (argdata.length+4);            /* length */
        data[3] = (byte) iCmd;                  /* command number */
        data[4] = (byte) iType;                 /* argument type */
        for (offset=0; offset<argdata.length; offset++) data[5+offset] = argdata[offset];

        int chksum = calc_chksum(data);   /* check sum */
        data[5+offset] = (byte)(chksum>>8);
        data[6+offset] = (byte)(chksum);

        return data;
}

private boolean connect(Socket sock) {
        byte data[]=null;
        byte argu[] = new byte[2];
        byte    indata[] = new byte[255];

        try {
                for (int i=0; i<3; i++) {
                        data = getSync(i);
                        output.write(data);

                        dispMsg("Write Sync" + i + "\n");

                        int     len;
                        int     iRet;

                        len = input.read(indata);
                        iRet = parseCmd(indata);
                        while (len==0 || iRet!=i) {
                                len = input.read(indata);
                                iRet = parseCmd(indata);
                        }
                }

                if (b_connected){
                        data = getCmd(1,0x3B,null);
                        output.write(data);
                        dispMsg("Open motor controller\n");

                        argu[0] = (byte)1;
                        argu[1] = (byte)0;
                        data = getCmd(4,0x3b,argu);             /* enable motor */
                        output.write(data);
                        dispMsg("Motor enabled\n");
                        return true;
                }
        }
        catch (Exception e) {
                dispMsg("Connect Error !!!\n");
                e.printStackTrace();
        }

        return false;
}
```

```
private int revShort(DataInputStream src) {
        int ret=0;

        try {
                byte buf1=src.readByte();
                byte buf2=src.readByte();

                ret = (int)0xff & buf1;
                ret |= (int)(buf2<<8);
        }catch(IOException e) {
                dispMsg("read reversed short error.\n");
        }
        return ret;
}

private float getPos(DataInputStream src) {
        int     buf;
        byte    buf1;
        int     buf2;
        float   ret;

        buf = revShort(src);
        buf1 = (byte)(buf>>8);
        buf2 = (int)0x00ff & buf;

        if ((buf1&(byte)0x40)!=0) {             //negative
                buf1 = (byte)(buf1 | (byte)0x80);
                buf = ((int)buf1)<<8 | (int)buf2;
        }

        ret = (float)(buf*param.DistConvFactor);
        return ret;
}


private String parseSIP(DataInputStream src) {
        String s_ret = "";

        try {
                //header part
                if (src.readByte()!=(byte)0xfa) return null;
                if (src.readByte()!=(byte)0xfb) return null;

                int len = src.readByte();                                       //counter

               switch (src.readByte()) {    /* status */
               case (byte)0x32:
                        s_ret += "Status: " + "stopped.\n";
                        motion = 0;
                        break;
               case (byte)0x33:
                        s_ret += "Status: " + "moving.\n";
                        motion = 1;
                        break;
               }
```

```
                    /* X, Y, Th position */
                    Xpos = getPos(src);
                    Ypos = getPos(src);
                    Thpos = (int)(revShort(src)*param.AngleConvFactor*180/Math.PI);

                    /* L & R velocity reading*/
                    float Lvel = (float)(revShort(src)*param.VelConvFactor);
                    float Rvel = (float)(revShort(src)*param.VelConvFactor);

                    float Battery = (float)src.readUnsignedByte()/10;          // battery charge in 10th
                    int bumpers = revShort(src);
          // motor stall & bumper accessory indicator
                    Control = (int)(revShort(src)*param.AngleConvFactor*180/Math.PI);      //angular
pos servo
                    int PTU = (src.readUnsignedByte()+src.readUnsignedByte()*256);
                    Compass = src.readUnsignedByte()*2;
                    s_ret += "Vel: " + Math.max((int)Lvel,(int)Rvel) + ".00 mm\n";
                    s_ret += "Rot: -2\n\n";

                    s_ret += "X: " + Xpos + "mm\n";
                    s_ret += "Y: " + Ypos + "mm\n";
                    s_ret += "Th: " + Thpos + "deg\n\n";

                    s_ret += "SPac: 20\n";
                    s_ret += "MPac: 18\n";
                    s_ret += "VPac: 0\n\n";

                    s_ret += "Batt: " + Battery +"V\n";
                    s_ret += "CPU: 0% \n";
                    s_ret += "Disp: 5.0Hz\n\n";

                    s_ret += "Comp: " + Compass + " \n";
                    for (int new_readings = src.readByte(); new_readings>0; new_readings --) {
                              int sonar_num = src.readByte();
                              float range =
(float)(src.readUnsignedByte()+256*src.readUnsignedByte())*param.RangeConvFactor;
                              sonar_read[sonar_num] = range;
//                            dispMsg("SONAR " + sonar_num + " reading: " + range + "mm. \n");
                    }
          } catch(IOException e) {
            dispMsg("IO error in parsing SIP.\n");
            return null;
          }

          return s_ret;
  }

  public boolean disconnect(Socket sock) {
            byte data[]=null;

            if (sock != null) {
                      try {
                                data = getCmd(2,0x3b,null);
                                output.write(data);
                                dispMsg("Close robot controller\n");
```

```
                        b_connected = false;
                        return true;
                }
                catch (Exception e) {
                        dispMsg("Disconnect Failed!!!\n");
                        e.printStackTrace();
                }
        }
        return false;
}

public void start() {
}

public void stop() {
}

public void run() {
        byte data[]=null;
        byte argu[] = new byte[2];
        byte indata[]=new byte[256];
        String s_status = null;
        Thread me = Thread.currentThread();

        while (t_pulse == me) {
                try {
                        Thread.currentThread().sleep(100);
                        data = getCmd(0,0x3b,null);          /*send pulse*/
                        output.write(data);
                        input.read(indata);
                        s_status = parseSIP(new DataInputStream(new
ByteArrayInputStream(indata)));

                        switch (status) {
                        case    1:      //turn
//                              if {
//
dispMsg("Control:"+Control+"degSet:"+degSet+"Thpos:"+Thpos+" \n");
                                if ((Control==Thpos) && (motion==0)){

        dispMsg("deg:"+degSet+"Thpos:"+Thpos+"turned.\n");
                                        argu[0] = (byte)0x3f;
                                        argu[1] = (byte)0x00;
                                        data = getCmd(0x0b, 0x3b, argu);
                                        output.write(data);
                                        status = 2;
                                }
                                break;
                        case 2: //translation
                                float range = getDiff();
                                if (range>f_Dist) {
                                        dispMsg("reached.\n");
                                        argu[0] = (byte)0x00;
                                        argu[1] = (byte)0x00;
                                        data = getCmd(0x0b, 0x3b, argu);      //E-Stop
```

```
                                        output.write(data);

                                        data = getCmd(7, 0x3b , argu);          //reset to origin
                                        output.write(data);
                                        tgtX = Xpos;
                                        tgtY = Ypos;
                                        f_Dist = 5000;
                                        status = 0;
                                }else {
                                        f_Dist = range;
                                        if (range<500) {
                                                argu[0] = (byte)(range/2);
                                                argu[1] = 0x00;
                                        } else {
                                                argu[0] = (byte)0x3f;
                                                argu[1] = (byte)0x00;
                                        }
                                        data = getCmd(0x0b, 0x3b, argu);
                                        output.write(data);
                                }
                                break;
                        default: //idle
                                argu[0] = (byte)0x00;
                                argu[1] = (byte)0x00;
                                data = getCmd(0x0b, 0x3b, argu);      //clear vel
                                output.write(data);
                                data = getCmd(7, 0x3b, argu);         //reset to origin
                                output.write(data);
                                tgtX = Xpos;
                                tgtY = Ypos;
                        }

//                      dispMsg("Get one SIP\n");
//                      dispMsg(s_status);
                        ta_Status.setText(s_status);
                } catch (Exception e) {
                        dispMsg("Thread Exception encountered...\n");
                        e.printStackTrace();
                }
                p_image.repaint();
        }
  }

  public float getDiff() {
        return (float) Math.sqrt((tgtX-Xpos)*(tgtX-Xpos)+(tgtY-Ypos)*(tgtY-Ypos));
  }
  public void destroy() {
    if (sock != null) {
                try {
                        sock.close();
                }
                catch (Exception e) {
                        dispMsg("Destroyed Failed!!!\n");
                        e.printStackTrace();
                }
    }
```

```
  removeAll();
 }

 private void makeStop() {
          byte data[]=null;
          data = getCmd (55,0x3b, null);
          return;
 }

 private void dispMsg(String msg) {
          ta_Cmd.appendText(msg);
 }

 public boolean action(Event ev, Object obj) {
  if (ev.target == bt_Conn) {
                  if (connect(sock)) {
                          if (t_pulse==null) {
                                  t_pulse = new Thread(this);
                                  t_pulse.suspend();

                                  t_pulse.start();
                          }
                          else t_pulse.resume();

                  }
   } else if (ev.target == bt_Disc) {
                          if (disconnect(sock)) t_pulse.suspend();
          } else if (ev.target == bt_Stop) {
                          makeStop();
          }
/*
          else if (ev.target == tf_Cmd) {
                  byte data[] = null;
                  byte argu[] = new byte[2];

                  try {
                          dispMsg("Get one command"+tf_Cmd.getText()+" \n");

                          argu[0] = (byte)3;
                          argu[1] = (byte)0;
                          data = getCmd(0x15,0x3b,argu);
                          output.write(data);
                          dispMsg("Command execed\n");
                  } catch(Exception e) {
                          dispMsg("rotate command");
                          e.printStackTrace();
                  }
          }
*/
          return true;
 }

 public boolean handleEvent(Event ev) {
  boolean rc = super.handleEvent(ev);

  return rc;
```

```
}

public int getRevX(float inX, float inY) {
        float factor = (float)(param.ImageRangeMax/((int)0x2bba*param.RangeConvFactor));
        float alpha = (float)(Math.PI/2-Thpos*Math.PI/180);
        return (int)(((inX-Xpos)*Math.cos(alpha)-(inY-Ypos)*Math.sin(alpha))*factor);
}

public int getRevY(float inX, float inY) {
        float factor = (float)(param.ImageRangeMax/((int)0x2bba*param.RangeConvFactor));
        float alpha = (float)(Math.PI/2-Thpos*Math.PI/180);
        return (int)(((inX-Xpos)*Math.sin(alpha)+(inY-Ypos)*Math.cos(alpha))*factor);
}

public float getAbsX(int inX, int inY) {
        float factor = (float)(((int)0x2bba*param.RangeConvFactor)/param.ImageRangeMax);
        float alpha = (float)(Thpos*Math.PI/180-Math.PI/2);
        return (float)((inX*Math.cos(alpha)-inY*Math.sin(alpha))*factor);
}

public float getAbsY(int inX, int inY) {
        float factor = (float)(((int)0x2bba*param.RangeConvFactor)/param.ImageRangeMax);
        float alpha = (float)(Thpos*Math.PI/180-Math.PI/2);
        return (float)((inX*Math.sin(alpha)+inY*Math.cos(alpha))*factor);
}

class ImagePanel extends Panel{
  private          Image          image;
          private Image          imgOfScrn;
          private Graphics   imgBuf;

  public ImagePanel(URL codebase, String imgFile) {
     image = getImage(codebase, imgFile);
                imgOfScrn = createImage(getSize().wi dth, getSize().height);
//              imgBuf = imgOfScrn.getGraphics();
  }

private void moveTo(int x, int y) {
        float xp, yp, range;
        byte    argu[] = new byte[2];
        byte    data[] = null;
        boolean         rel = true;

        try{
                dispMsg("X:"+x+"Y:"+y+" \n");
                x -= param.ImageCenterX;
                y = param.ImageCenterY-y;

                degSet = (int)((Math.atan2(y,x)*180/Math.PI)-90);
                dispMsg("Compass:" + Compass +"rel deg: "+degSet +" \n");
                if (rel) {
                        int deg = degSet<-180?degSet+360:degSet;
                        byte flag = deg>0?(byte)0x3b:(byte)0x1 b;
                        argu[0] = (byte)(Math.abs(deg));
                        argu[1] = (byte)0x00;
                        data = getCmd(0x0d,flag,argu);
```

```
                } else {
                        argu[1] = (byte)(degSet>255?0x01:0x00);
                        argu[0] = (byte)(degSet>255?degSet-255:degSet);
                        data = getCmd((byte)0x0c,0x3b,argu);
                }
                output.write(data);
                degSet += Compass;
                degSet = degSet<0?degSet+360:(degSet>360?degSet -360:degSet);
                dispMsg("Abs deg: "+degSet+" \n");
                tgtX = getAbsX(x,y);
                tgtY = getAbsY(x,y);

                dispMsg("Target X:" + tgtX + "Target Y:" +tgtY+" \n");
                status = 1;
        }catch(IOException e) {
                 dispMsg("move to error\n");
        }
}

  public boolean mouseUp(Event evt, int x, int y)
  {
                moveTo(x,y);

                return true;
  }

        public void updateBuf(Graphics g) {
    //Draw image at its natural size first.
                int             x,y;
                float   range;

                g.drawImage(image,0,20,this);
                g.setColor(Color.magenta);
                for (int i=0; i<16; i++) {
                        range =
sonar_read[i]*param.ImageRangeMax/((int)0x2bba*param.RangeConvFactor);
                        if (range<param.ImageRangeMin || range >param.ImageRangeM ax)
continue;
                        x =
(int)(range*Math.cos(param.SonarUnitTh[i]*Math.PI/180+Math.PI/2));
                        y =
(int)(range*Math.sin(param.SonarUnitTh[i]*Math.PI/180+Math.PI/2));
                        g.drawRect(param.ImageCenterX+x, param.ImageCenterY+y,
param.ObjSize, param.ObjSize);
                }
                x = getRevX(tgtX, tgtY);
                y = getRevY(tgtX, tgtY);

                g.drawLine(x+param.ImageCenterX, param.ImageCenterY -y,
param.ImageCenterX, param.ImageCenterY);
        }

  public void paint(Graphics g) {
                updateBuf(g);
//              g.drawImage(imgOfScrn, 0, 0, this);
  }
```

```
}
        public class pPTZ extends Panel {
//                  DataOutputStream        output;
                    Button                          bt_up, bt_dn, bt_lf, bt_rt, bt_hm;
                    TextField                       tf_zm;

                    private final double        DEG_TO_PAN=9.26;
                    private final int   MAX_PAN = 95;
                    private final int   DEG_TO_TILT=15;
                    private final int   MAX_TILT = 20;
                    private final int   COMPTZCAM = 42;

                    private byte initb[] = {0x09,(byte)0x88,0x01,0x00,0x01,(byte)0xff,(byte)0x88,

0x30,0x01,(byte)0xff};
                    private byte zoomb[] = {0x09,(byte)0x81,0x01,0x04,0x47,0x00,0x00,

0x00,0x00,(byte)0xff};
                    private byte ptb[] = {0x0f,(byte)0x81,0x01,0x06,0x02,0x18.0x14,

0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,(byte)0xff};
                    private int PTZCamPanAngle = 0;
                    private int PTZCamTiltAngle = 0;

                    public pPTZ() {
                            Panel   p;

                            setLayout(new BorderLayout());
                            p = new Panel();
                            p.setLayout(new BorderLayout());
                            bt_up = new Button("Up");
                            p.add(bt_up,BorderLayout.NORTH);
                            bt_dn = new Button("Down");
                            p.add(bt_dn, BorderLayout.SOUTH);
                            bt_lf = new Button("Left");
                            p.add(bt_lf, BorderLayout.WEST);
                            bt_rt = new Button("Right");
                            p.add(bt_rt, BorderLayout.EAST);
                            bt_hm = new Button("Home");
                            p.add(bt_hm, BorderLayout.CENTER);
                            add(p, BorderLayout.CENTER);

                            p = new Panel();
                            p.setLayout(new FlowLayout());
                            p.add(new Label("Zoom (0-1023):"));
                            tf_zm = new TextField("0",4);
                            p.add(tf_zm, BorderLayout.CENTER);
                            add(p, BorderLayout.SOUTH);
//                          this.output = stream;
//                          this.PTZCamTilt(20);
                    }

                    private void setNibbles(int p os, byte []buf, int offset) {
                            buf[offset+1] = (byte)((pos & 0xf000)>>12);
```

```
        buf[offset+2] = (byte)((pos & 0x0f00)>>8);
        buf[offset+3] = (byte)((pos & 0x00f0)>>4);
        buf[offset+4] = (byte)((pos & 0x000f)>>0);
}

public void PTZInit() {
        byte data[]=null;

        try {
                PTZCamPanAngle = PTZCamTiltAngle = 0;
                setNibbles(0, ptb, 6);
                setNibbles(0, ptb, 10);
                data = getCmd(0x2a,0x2b,initb);
                output.write(data);
                setNibbles(0, zoomb, 4);
                output.write(data);
        }catch(IOException e) {
                dispMsg("init error\n");
        }
}

private void PTZCamPan(int deg) {
        int pos;
        byte data[] = null;

        try {
                if (deg>MAX_PAN) deg = MAX_PAN;
                if (deg<-MAX_PAN) deg = -MAX_PAN;
                pos = (int)(((double)deg)*DEG_TO_PAN);
                setNibbles(pos, ptb, 6);
                data = getCmd(0x2a,0x2b,ptb);
                output.write(data);
                this.PTZCamPanAngle = deg;
        }catch(IOException e) {
                dispMsg("pan error\n");
        }
}

private void PTZCamTilt(int deg) {
        int pos;
        byte data[] = null;

        try {
                if (deg>MAX_TILT) deg = MAX_TILT;
                if (deg<-MAX_TILT) deg = -MAX_TILT;
                pos = (int)(((double)deg)*DEG_TO_TILT);
                setNibbles(pos, ptb, 10);
                data = getCmd(0x2a,0x2b,ptb);
                output.write(data);
                this.PTZCamTiltAngle = deg;
        }catch(IOException e) {
                dispMsg("tilt error\n");
        }
}

private void PTZCamPanTilt(int pan, int tilt) {
```

```
        int             pos;
        byte    data[] = null;

        try {
                if (tilt>this.MAX_TILT) tilt = this.MAX_TILT;
                if (tilt<-this.MAX_TILT) tilt = -this.MAX_TILT;
                if (pan>this.MAX_PAN) pan = this.MAX_PAN;
                if (pan<-this.MAX_PAN) pan = -this.MAX_PAN;

                pos = (int)(((double)pan)*this.DEG_TO_PAN);
                this.setNibbles(pos, ptb, 6);
                pos = (int)(((double)tilt)*this.DEG_TO_TILT);
                this.setNibbles(pos, ptb, 10);
                data = getCmd(0x2a,0x2b,ptb);
                output.write(data);
                this.PTZCamPanAngle = pan;
                this.PTZCamTiltAngle = tilt;
        }catch(IOException e) {
                dispMsg("pan-tilt error\n");
        }
}

private void PTZCamZoom(int val) {
        byte    data[]=null;
        try {
                if (val<0) val = 0;
                if (val > 1023) val = 1023;
                this.setNibbles(val, zoomb, 4);
                data = getCmd(0x2a, 0x2b, zoomb);
                output.write(data);
        }catch(IOException e) {
                dispMsg("zoom error\n");
        }
}

public boolean action(Event ev, Object obj) {
        if (ev.target == bt_up) {
                dispMsg("Up btn clicked\n");
                this.PTZCamTilt(this.PTZCamTiltAngle+2);
        }
        if (ev.target == bt_dn) {
                this.PTZCamTilt(this.PTZCamTiltAngle -2);
        }
        if (ev.target == bt_lf) {
                this.PTZCamPan(this.PTZCamPanAngle -2);
        }
        if (ev.target == bt_rt) {
                this.PTZCamPan(this.PTZCamPanAngle+2);
        }
        if (ev.target == bt_hm) {
                this.PTZCamPanTilt(0,0);
                this.PTZCamZoom(0);
        }
        if (ev.target == tf_zm) {
                this.PTZCamZoom(Integer.parseInt(tf_zm.getText()));
        }
```

```
                return true;
            }
        }
    }
```

```
#include <stdio.h>                    /* printf */
#include <fcntl.h>
#include <unistd.h>                   /* read */
#include <errno.h>                    /* errno */
#include <sys/mman.h>
#include <sys/time.h>
#include <stdlib.h>
#include "ioctl_meteor.h"
#include "ioctl_bt848.h"

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/* use bt848fg0 for 1st frame grabber
   use bt848fg1 for 2nd frame grabber */
#define VIDEO_DEV      "/dev/bt848"

/* a 1/4 sized NTSC frame */
#define COLS 320                      /* # of pixels in a row in output image */
#define ROWS 240                      /* # of lines in output image */
#define SIZE (ROWS * COLS * 4)        /* 4 bytes/pixel for METEOR_GEO_RGB24 */

/* define the following for the S-video input instead of the BNC composite */
/* USE_SVIDEO_INPUT */

#define BUFFERMAX      8192
#define SWAPBUFSIZE    230400

int formPPM(char *buf, char *out);
int  openSocket(int port);

int
main (int argc, char *argv[])
{
 struct meteor_counts cnt;
 struct meteor_geomet geo;
 char *buf, header[16], *p;
 int fd, c, j, k;
 char *mmbuf;

 char fn[20];
 struct timeval t;
 long t0, t1;
 int ic;   /* image file counter */

 int sock, fromlen, n;
 int port = 8104;
 struct sockaddr_in from;
 char lwf_buf[BUFFERMAX];
 char swap_buf[SWAPBUFSIZE];
 long lSize=0;

 int xOffset,yOffset,xSize,ySize;
```

```
sock = openSocket(port);
fromlen = sizeof(struct sockaddr_in);

/* open the device */
if ((fd = open(VIDEO_DEV, O_RDWR)) <= 0) {
  perror (VIDEO_DEV);
  exit (1);
}

/* initialize the Imagenation PXC200.
 *
 * WARNING: this function may hang Linux if the board is not a PXC200!!!
 */
printf ("initializing PXC200\n");
if (ioctl (fd, BT848_INITPXC200, &c) < 0) {
  perror ("BT848_INITPXC200 ioctl\n");
  exit (1);
}

/* set capture geometry */
geo.rows = ROWS;                    /* # of lines in output image */
geo.columns = COLS;                 /* # of pixels in a row in output image */
geo.frames = 1;             /* # of frames in a buffer */
geo.oformat = METEOR_GEO _RGB24 ; /* RGB 24 in 4 bytes: NULL,R,G,B */
if (ioctl (fd, METEORSETGEO, &geo) < 0) {
  perror ("METEORSETGEO ioctl\n");
  exit (1);
}

if (NULL == (buf = malloc(COLS*ROWS*4))) {
  perror ("malloc");
  exit (1);
}


/* set input video format */
c = METEOR_FMT_NTSC;
if (ioctl (fd, METEORSFMT, &c) < 0) {
  perror ("METEORSFMT ioctl\n");
  exit (1);
}

/* set input port */
#ifdef USE_SVIDEO_INPUT
  c = METEOR_INPUT_DEV0;        /* PXC200 S-video connector */
#else
  c = METEOR_INPUT_DEV1;        /* PXC200 BNC composite video connector */
#endif
  if (ioctl (fd, METEORSINPUT, &c) < 0) {
    printf ("METEORSINPUT ioctl failed: %d\n", errno);
    exit (1);
  }

  /* why do we need this here ! */
  usleep (600000);
```

```
/* memory map the frame grabber data buffer */
if (0 > (mmbuf = (char *) mmap((caddr_t) 0, ROWS*COLS*4, PROT_READ,
                    MAP_FILE | MAP_PRIVATE, fd, (off_t) 0))) {
 perror ("mmap failed");
 exit (1);
}

c = METEOR_CAP_CONTINOUS;
if (ioctl (fd, METEORCAPTUR, &c)) {
 perror ("ioctl ContiCapture failed");
 exit (1);
}
buf = mmbuf;

/* get error counts */
if (ioctl (fd, METEORGCOUNT, &cnt)) {
   perror("ioctl GetCount failed");
   exit (1);
}

/*
 *  MAIN LOOP
 */
printf("Start\n");

gettimeofday(&t, NULL);
t0 = t.tv_sec*1000 + t.tv_usec/1000;
t1 = t0;

for (ic=0; t1-t0<500000; ic++)    /* Timeout = 500 sec. */
{
    usleep(100);
          gettimeofday(&t, NULL);
          t1 = t.tv_sec*1000 + t.tv_usec/1000;

          /* save the image as a PPM file */
    n = recvfrom(sock,buf,1024,0,(struct sockaddr *)&from,&fromlen);
    if (n < 0) {
          perror("recvfrom error");
          exit(0);
     }
          formPPM(buf,swap_buf);
    write(1,"Received a datagram: ",21);
    write(1,lwf_buf,n);
     sscanf(lwf_buf, "%d,%d,%d,%d",&xOffset,&yOffset,&xSize,&ySize);
     lSize = saveLWF(lwf_buf, ROWS, COLS, xOffset, yOffset, xSize, ySize, swap_buf);

    n = sendto(sock,lwf_buf,lSize,0,(struct sockaddr *)&from,fromlen);
    if (n  < 0) {
          perror("sendto error");
          exit(0);
     }

}
gettimeofday(&t, NULL);
t1 = t.tv_sec*1000 + t.tv_usec/1000;
```

```
 c = METEOR_CAP_STOP_CONT;
 if (ioctl(fd, METEORCAPTUR, &c)) {
        perror("ioctl SingleCapture2 failed");
        exit(1);
 }

 close (fd);                        /* close the capture device */
}

int formPPM(char *buf, char *out) {
        char *in = buf;
        char *rgb=out;
        int     k,j;

        for (j=0; j<ROWS; j++) {
          for (k=0; k<COLS; k++) {
                rgb[2] = *in++;    /* blue */
                rgb[1] = *in++; /* green */
                rgb[0] = *in++;    /* red */
                in++;              /* NULL byte */
                rgb += 3;
           }
          }

        return 0;
}

int openSocket(int port) {
  int sock, length;
  struct sockaddr_in server;

  sock=socket(AF_INET, SOCK_DGRAM, 0);
  if (sock < 0) {
     perror("Opening socket");
     exit(0);
  } else perror("Open socket OK");

  length = sizeof(server);
  bzero(&server,length);
  server.sin_family=AF_INET;
  server.sin_addr.s_addr=INADDR_ANY;
  server.sin_port=htons(port);
  if (bind(sock,(struct sockaddr *)&server,length)<0) {
     perror("binding");
     exit(0);
  }

  return sock;
}
```

```
import java.awt.*;
import java.awt.image.*;
import java.util.*;
import Vector3D;
import Raster;

public class PlanarView
{
        protected Vector3D O;       //centre of projection
        protected Matrix3x3 P;      //projection manifold matrix
        protected boolean invFlag;
        protected Matrix3x3 iP;     //inverse matrix

        public PlanarView(){
                O = new Vector3D(0,0,0);
                P = new Matrix3x3();
                invFlag = false;
        }

        public PlanarView(Vector3D O, Matrix3x3 P) {
                this.O = O;
                this.P = P;
                invFlag = false;
        }

        public PlanarView(PlanarView v){
                O = v.O;
                P = v.P;
                invFlag = false;
        }

        public PlanarView(Vector3D eye, Vector3D look, Vector3D up, float hfov,  Dimension
win) {
                O = new Vector3D(eye);
                Vector3D Du = Vector3D.normalize(look.cross(up));
                Vector3D Dv = Vector3D.normalize(look.cross(Du));

                float fl = (float)(win.width / (2*Math.tan(0.5*hfov)));
                Vector3D Vp = Vector3D.normalize(look);
                Vp.x = Vp.x*fl - 0.5f*(win.width*Du.x + win.height*Dv.x);
                Vp.y = Vp.y*fl - 0.5f*(win.width*Du.y + win.height*Dv.y);
                Vp.z = Vp.z*fl - 0.5f*(win.width*Du.z + win.height*Dv.z);
                P = new Matrix3x3(Du, Dv, Vp);
                invFlag = false;
        }

        public PlanarView clone(PlanarView original) {
                return new PlanarView(new Vector3D(original.O), new Matrix3x3(original.P));
        }

        public final Vector3D baseline(PlanarView V) {
                return new Vector3D(O.x  - V.O.x, O.y - V.O.y, O.z - V.O.z);
        }

        public final Vector3D inverseMap(Vector3D V) {
                if (invFlag == false) {
```

```
                    iP = P.inverse();
                    invFlag = true;
                }

                return iP.times(V);
        }

        public final Vector3D origin() {
                return new Vector3D(O.x, O.y, O.z);
        }

        public final Vector3D Du() {
                return new Vector3D(P.element(0,0), P.e lement(1,0), P.element(2,0));
        }

        public final Vector3D Dv() {
                return new Vector3D(P.element(0,1), P.element(1,1), P.element(2,1));
        }

        public final Vector3D Vp() {
                return new Vector3D(P.element(0,2), P.element(1,2), P.element(2,2));
        }

        public Vector3D map(int u, int v) {
                return new Vector3D(P.M[0]*u + P.M[1]*v + P.M[2],
                                                        P.M[3]*u + P.M[4]*v + P.M[5],
                                                        P.M[6]*u + P.M[7]*v + P.M[8]);

        }
}
```

```
import java.awt.*;
import java.awt.image.*;
import java.util.*;

public class Raster implemen ts ImageObserver
{
        public int width, height;
        public int pixel[];
        private ImageProducer producer;

        public Raster(){}

        public Raster(int w, int h) {
                width = w;
                height = h;
                pixel = new int[w*h];
        }

        public Raster(Image img) {
                width = -1;
                height = -1;
                int w = img.getWidth(this);
                int h = img.getHeight(this);
                if (w>=0) width = w;
                if (h>=0) height = h;

                try {
                        while ((width<0)||(height<0)) {
                                Thread.sleep((long) 100);
                        }
                        if (width*height==0) return;
                        pixel = new int[width*height];
                        PixelGrabber pg = new
PixelGrabber(img,0,0,width,height,pixel,0,width);
                        pg.grabPixels();
                }catch (InterruptedException e) {
                        width = 0;
                        height = 0;
                        pixel = null;
                        return;
                }
        }

        public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
                if ((flags & WIDTH) !=0) width = w;
                if ((flags & HEIGHT) !=0) height = h;
                if ((flags & ABORT) !=0) { width=0; height = 0;}
                if ((flags & ERROR) !=0) { width = 0; height = 0;}
                return ((flags & ALLBITS) !=0);
        }

        public final int size() {
                return width*height;
        }

        public final void fill(Color c) {
```

```
                int s = size();
                int rgb = c.getRGB();
                for (int i=0;i<s;i++)
                        pixel[i] = rgb;
        }

        public Image toImage(Component root) {
                return root.createImage(new MemoryImag eSource(width, height, pixel,
0,width));
        }

        public final int getPixel(int x, int y) {
                return pixel[y*width+x];
        }

        public final Color getColor(int x, int y) {
                return new Color(pixel[y*width+x]);
        }

        public final boolean setPixel(int pix, int x, int y) {
                pixel[y*width+x] = pix;
                return true;
        }

        public final boolean setColor(Color c, int x, int y) {
                pixel[y*width+x] = c.getRGB();
                return true;
        }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#include "readpnm.h"

int ReadPNMHeader(FILE * f, int * W, int * H,
            int * dx, int * dy,
            char ** comment,int *bin);

int  ReadPNM(char * FileName, unsigned char ** pixel,
         int * W, int * H,
         int * dx, int * dy,
         char ** comment)
{
     FILE * f;
                     int bin;
     if(  FileName==NULL || pixel==NULL
      ||      W==NULL ||    H==NULL
      ||     dx==NULL ||   dy==NULL)
       return -1;

     f=fopen(FileName,"rb");
     if(f==NULL) return -2;

     if(ReadPNMHeader(f, W,  H, dx, dy, comment, &bin)<0)return  -3;
                     if(!bin) return -4;

     (*pixel)=(unsigned char*)calloc(*W * *H* *dx, sizeof(unsigned char));
     if(*pixel==NULL) return -4;

     if(fread( *pixel, sizeof(unsigned char), *W**H**dx, f)!=(unsigned) (*W**H**dx) ) {
          fclose(f);
          if(comment)
     if(*comment) free(*comment);
          free(*pixel);
          *pixel=NULL;
          return -5;
     }

     fclose(f);
           return 0;
}


int readInt(FILE * f,char **pcomment, int * error, char * BeginComment,
         int MaxComment)
{
     int IsComment=0;       int InNr=0;
     char c;
     char buf[6];
     char *b=buf;

     while(fread(&c,1,1,f)==1){
      if(IsComment){
```

```
            if(*pcomment-BeginComment<MaxComment){
                **pcomment=c;
                (*pcomment)++;
            }
            if(c=='\n')IsComment=0;
            else ;
        }
        else{
            if(InNr)
                if(isdigit(c))*b++=c;
                else {
                  *b++=0;
                  fseek(f,-1,SEEK_CUR);
                  return atoi(buf);
                }
            else
                if(isdigit(c)){
                  *b++=c;
                  InNr=1;
                }
                else
                if(c=='#')IsComment=1;
                else
                  if(!isspace(c)){
                        *error=1;
                        return 0;
                  }
        }
             }
      return atoi(buf);
}

int ReadPNMHeader(FILE * f, int * W, int * H,
             int * dx, int * dy,
             char ** comment, int * bin)
{
 char MagicNr[3]="";
 char * Comment,c;
 char *pcomment;
 int maxv;
 int error;

 error=0;
 if((Comment=(char *)calloc(MAXCOMMENT,sizeof(char)))==NULL){
      fprintf(stderr,"no memory\n");
      return -1;
 }
 pcomment=Comment;
 fread(MagicNr,2,1,f);MagicNr[2]=0;

 *bin=0;
 if(!strcmp(MagicNr,"P6")){*dx=3;*bin=1;}      /*PPM -BIN*/
 else if(!strcmp(MagicNr,"P5")){*dx=1;*bin=1;}  /*PGM -BIN*/
 else if(!strcmp(MagicNr,"P2"))*dx=1;          /*PGM -ASCII*/
 else if(!strcmp(MagicNr,"P3"))*dx=3;          /*PPM -ASCII*/
 else  {
```

```
            free(Comment);
            return -1;
  }
  *W=readInt(f,&pcomment,&error, Comment,MAXCOMMENT);  /* width */
  if(error) return -1;
  *dy=*W * *dx;
  *H=readInt(f,&pcomment,&error, Comment,MAXCOMMENT);  /* height  */
  if(error) return -1;
  maxv=readInt(f,&pcomment,&error,Comment,MAXCOMMENT);
  if(error) return -1;
  if(fread(&c,1,1,f)!=1){
            free(Comment);
            return -1;
      }
  if(!isspace(c)){
            free(Comment);
             return -1;
             }
  *pcomment=0;
  if(comment!=NULL)*comment=s trdup(Comment);
  free(Comment);
  return 0;
}


int ReadPNMStripe(FILE *f, unsigned char * stripe,
             int W, int H, int nchannels,
             int dx, int dy,int bin)
{
            unsigned char * line, *pix, *dot;
            int y,x,c;
            if(f==NULL || stripe==NULL) return -1;
            if(nchannels!=1 && nchannels!=3) return -2;

            if(bin)
                        for(line=stripe,y=H;y--; line+=dy)
                                    for(pix=line,x=W;x--; pix+=dx){
                                                if(fread(pix,nchannels,1,f)!=1) return -1;}
            else
                        for(line=stripe,y=H;y--; line+=dy)
                                    for(pix=line,x=W;x--; pix+=dx)
                                                for(dot=pix,c=nchannels;c--;)
                                                fscanf(f,"%3d",dot++);
            return 0;
}
```

```
import java.awt.*;
import java.awt.image.*;
import Raster;

public class Reference extends Raster
{
        protected float maxDisparity, minDisparity;

        //constructor
        public Reference(Component root, Image img, Image dsp, float maxd, float mind) {
                super(img);                     //get the image pixels
                Raster dspras = new Raster(dsp);     //get the disparity values

                //determine the pixels buffer's width and height
                if ((dspras.width != width) || (dspras.height != height)) {
                        System.err.println("Error: image and disparity are of different sizes");
                        return;
                }

                //copy the disparity information into the pixel array

                for (int p=0; p<pixel.length; p++) {
                        int rgb = pixel[p] & 0x00ffffff;
                        int d = dspras.pixel[p] & 0x0000ff00;
                        pixel[p] = (d << 16) | rgb;
                }

                maxDisparity = maxd;
                minDisparity = mind;
        }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "savepnm.h"

/*
 *
 *  definition of the function SavePNM
 */

int WritePNM(FILE * f, char * Comment,
                                        int W, int H,
                                        int dx, int dy,
                                        int nchannels,
                                        char * pixel)
{
        char * buf;
        int MgNr;

        if(f==NULL || pixel==NULL) return -3;
        if(W<=0 || H<=0 || dx==0 || dy==0) return -4;
        if(nchannels!=1 && nchannels!=3) return -5;

        MgNr=(nchannels == 1  ?  5 : 6 );
        if(Comment!=NULL){
                buf=(char*)calloc(strlen(Comment)+50, sizeof(char));
                if(buf==NULL) {
                                return -7;
                }
                sprintf(buf,"P%1d%c# %s%c%d %d%c255%c",

       MgNr,10,Comment,10,W,H,10,10);
        }
        else{
                buf=(char*)calloc(50, sizeof(char));
                if(buf==NULL){
                                return -7;
                }
                sprintf(buf,"P%1d%c%d %d%c255%c",
                                                        MgNr,10,W,H,10,10);
        }

        /* write header */
        if(fwrite(buf, strlen(buf), 1, f)!=1) return -8;
        /* write data */
        if(dx==nchannels){
                /* no "alpha"-channels */
                if(fwrite(pixel, W*H*nchannels, 1, f)!=1) return -9;
          else;
        }
        else{
                char * line, *pix, *dot;
                int z,s,c;

                /* with "alpha"-channels */
```

```
                for(z=0, line=pixel; z<H; z++, line+=dy)
                  for(s=0, pix=line; s<W; s++, pix +=dx)
                          for(c=0, dot=pix; c<nchannels;c++)
                    fputc(*dot++, f);

        }


        if(buf!=NULL)free(buf);

        return 0;

}

int SavePNM(char * Filename,
                                        char * Comment,
                                        int W,int H,
                                        int dx, int dy,
                                        int nchannels,
                                        char * pixel)
{
         FILE * f;
         int r;

         if(Filename==NULL) return -1;
         f=fopen(Filename,"wb");
         if(f==NULL) return -2;
         r=WritePNM( f, Comment, W, H, dx, dy, nchannels, pixel);
         fclose(f);
         return r;
}

int WritePNMHeader(FILE * f, int W, int H, int nchannels, int dx, int dy,char * comment, int bin)
{
        int Mg;
        if(f==NULL) return -1;

        if(bin) Mg=(nchannels==1 ? 5 : 6);
        else    Mg=(nchannels==1 ? 2 : 3);

        if(comment==NULL)
                fprintf(f,"P%1d%c%d %d%c255%c",Mg,10,W,H,10,10);
        else
                fprintf(f,"P%1d%c# %s%c%d %d%c255%c",Mg,10,comment,10,W,H,10,10);

        return 0;
}

int WritePNMStripe(FILE * f, unsigned char * stripe,
                                        int W, int H, int nchannels,
                                        int dx, int dy,int bin)
{
        unsigned char * line,*pix,*dot;
        int y,x,c;
        if(nchannels>3 || nchannels<1) return -1;
        if(bin)
```

```
        for(y=H,line=stripe;y--;line+=dy)
                for(pix=line,x=W;x--;pix+=dx){
                        if(fwrite(pix, nchannels, 1,f)!=1) return -2;
        }
        else
        for(y=H,line=stripe;y--;line+=dy){
                for(pix=line,x=W;x--;pix+=dx)
                        for(dot=pix,c=nchannels;c--;dot++)
                          fprintf(f,"%4d", *dot);
                fprintf(f,"%c",10);
        }
        return 0;
}
```

```
import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.image.*;
import java.applet.Applet;

public class sock extends Applet implements Runnable{
 int                port;
 String   host;

 DatagramSocket socket = null;
 DatagramPacket inPack = null;
 DatagramPacket outPack = null;
 byte buffer[] = new byte[8192];
 InetAddress serverAddress = null;

 Label              displayLabel;
 String   s_Query;

 lwfDecoder        imageDecoder;
 private final String LICENSE         = "045302-2293052";
 private final String PARAM_src               = "src";  // The name used to define the URL
 private final String PARAM_width             = "width";          // The name used t o define the
width of the

                    // image displayed inside the applet's window

                    // In the most cases equal to <APPLET WIDTH=...>
 private final String PARAM_height            = "height";         // Respective the height
 private final String PARAM_limit   = "limit";            // Number of bytes processed for
decompression

                    // "Reconstruction rate"

                    // This amount of data is downloaded and only

                    // the data corresponding that limit is decompressed
 private final String PARAM_scale  = "scale";            // Scaling factor for display

                    // 1 means downscale to half-size

                    // 2 means downscale to quarter-size

 private final String PARAM_password = "pass";      // Passwort
 private final String PARAM_interval = "interval";     //image grab interval

 private URL                m_url= null;
 private Image              m_Image = null;
 private String     m_Password = null;
 private int                m_limit = -1;
 private int                m_scale  = 0;

 private boolean  m_Error = false;
 private String     m_ErrorMessage = "";

 private int                xOffset=0, yOffset=0, xSize=0, ySize=0;
```

```
private int                 newX=0, newY=0;
private int                 interval;

private Thread   t_This;

public synchronized void init() {
        int                 index,imagesize;
        int                 scale;
        String    param;
        DataInputStream stream;

 host = getParameter("host");
 if (host == null) {
  host = "129.128.68.89";
 }

 String Port = getParameter("port");
 if (Port == null) {
  port = 8104;
 } else {
  port = Integer.parseInt(Port);
 }

        String Inter = getParameter("interval");
        if (Inter==null) interval = 1000;
        else interval = Integer.parseInt(Inter);

        // Reading the Applet-Parameter
        param = getParameter(PARAM_limit);
        m_limit = param != null ? Integer.parseInt(param) : 0;

        param = getParameter(PARAM_sca le);
        m_scale = param != null ? Integer.parseInt(param) : 0;

        param = getParameter(PARAM_password);
        if (param != null) m_Password = param;

        displayLabel = new Label("  Interest Area Position  ");
        displayLabel.setAlignment(Label.CENTER);
        add(displayLabel);
try {
  dispMsg("\nOpening socket: ");
  socket = new DatagramSocket(port);
  serverAddress = InetAddress.getByName(host);

          stream = grabImage("0,0,-1,-1\n");
          createImage(stream);
}
catch (Exception e) {
  dispMsg("Open Failed !!!");
  e.printStackTrace();
          return;
}

        t_This = new Thread(this);
        t_This.suspend();
```

```
        t_This.start();
    }

    private DataInputStream grabImage(String cmd) {
        DataInputStream  stream;

        try{
            cmd.getBytes(0, cmd.length(), buffer, 0);
            outPack = new DatagramPacket(buffer, cmd.length(), serverAddress, port);
            socket.send(outPack);
            dispMsg("Load Image...");

            inPack = new DatagramPacket(buffer, buffer.length);
            socket.receive(inPack);
            buffer = inPack.getData();
            stream = new DataInputStream(new ByteArrayInputStream(buffer));
            dispMsg("Get Image...");
        } catch(Exception e) {
            dispMsg("Grab Image Failed !!!");
            e.printStackTrace();
            return null;
        }
        return stream;
    }

    private void createImage(DataInputStream stream) {
        int             image[];
        int             w,h;

        try {
            imageDecoder    = new lwfDecoder(stream,m_Password,LICENSE);

            // Creation of the corresponding buffer of pixels
            w                           = imageDecoder.getWidth() >> m_scale;
            h                           = imageDecoder.getHeight() >> m_scale;
            image                   = new int[w * h];

            // Decoding into memory using the parameters given at creation date
            imageDecoder.decodeToMemory(image,m_limit,1024,m_scale);

            // Blitting the image buffer using a Memory -Image Source
            m_Image         = createImage(new MemoryImageSource(w,h,image,0,w));

            repaint();
        }catch (Exception e) {
            m_Error = true;
            e.printStackTrace();
            m_ErrorMessage = e.getMessage();
        }
        return;
    }

    private void dispMsg(String msg) {
        displayLabel.setText(msg);
//      System.out.println(msg);
    }
```

```
// APPLET-INFO-SUPPORT
public String getAppletInfo()
{
        return "Name: LuraWave Decoder Applet \r\n" +
                        "(c) 2000 LuraTech GmbH";
}

// PARAMETER-INFO-SUPPORT
public String[][] getParameterInfo()
{
        String[][] info = {
                        { PARAM_src,            "String","Image URL" },
                        { PARAM_limit,          "Integer",      "Amount of bytes to
display" },
                        { PARAM_scale,          "Integer",      "Skaling of the image"},
                        { PARAM_interval,       "Integer",      "Image grabbing
interval"}
        };
        return info;
}

public void paint(Graphics g)
{
        if (m_Image != null) {
                g.drawImage(m_Image,0,0,null);
                if (xSize>0 && ySize>0) {
                        g.setColor(Color.white);
                        g.drawRect(xOffset, yOffset, xSize, ySize);
                }
        } else {
//              g.drawString("LuraWave Decoder Applet", 10, 20);
//              g.drawString("(c) 2000 LuraTech GmbH", 10, 35);
                if (m_Error) g.drawString("Applet Error: " + m_ErrorMessage, 10, 50);
        }
}

public boolean mouseDown(Event evt, int x, int y)
{
        newX = x;
        newY = y;
//      dispMsg("clicked!");

        return true;
}

public boolean mouseUp(Event evt, int x, int y) {
        DataInputStream         stream;
        if(evt.modifiers ==0) {     //left button
                xSize = Math.abs(x - newX);
                ySize = Math.abs(y - newY);
                xOffset = Math.min(x, newX);
                yOffset = Math.min(y , newY);
        } else {
                xOffset = x;
                yOffset = y;
```

```
                xSize = -1;
                ySize = -1;
        }
        dispMsg("released");

        return true;
}

public void start(){
        t_This.resume();
}

public void stop(){
        t_This.suspend();
}

public void destroy() {
        t_This.stop();
}

public void run() {
        DataInputStream         stream;
        String                  s_cmd;

        Thread me = Thread.currentThread();
        while(t_This == me) {
                try {
                        Thread.sleep(interval);
                        s_cmd = ""+xOffset+","+yOffset+","+xSize+","+ySi ze+"\n";
                        stream = grabImage(s_cmd);
                        createImage(stream);
                } catch(Exception e) {
                        dispMsg(e.toString());
                }
        }
 }

}
```

```
import java.util.*;
import Matrix3x3;

public class Vector3D
{
	public float x, y, z;
	public Vector3D(){}

	public Vector3D(float x, float y, float z) {
		this.x = x;
		this.y = y;
		this.z = z;
	}

	public Vector3D(StringTokenizer parser) throws NumberFormatException {
		x = Float.valueOf(parser.nextToken()).floatValue();
		y = Float.valueOf(parser.nextToken()).floatValu e();
		z = Float.valueOf(parser.nextToken()).floatValue();
	}

	public Vector3D(Vector3D v) {
		x = v.x;
		y = v.y;
		z = v.z;
	}

	//methods
	public final float dot( Vector3D B) {
		return (x*B.x+y*B.y+z*B.z);
	}

	public final float dot(float Bx, float By, float Bz) {
		return (x*Bx+y*By+z*Bz);
	}

	public static final float dot( Vector3D A,  Vector3D B) {
		return (A.x*B.x + A.y*B.y + A.z*B.z);
	}

	public final Vector3D cross( Vector3D B) {
		return new Vector3D(y*B.z-z*B.y, z*B.x-x*B.z, x*B.y-y*B.x);
	}

	public final Vector3D cross(float Bx, float By, float Bz) {
		return new Vector3D(y*Bz-z*By, z*Bx-x*Bz, x*By-y*Bx);
	}

	public final static Vector3D cross( Vector3D A,  Vector3D B) {
		return new Vector3D(A.y*B.z-A.z*B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x);
	}

	public final float length() {
		return (float) Math.sqrt(x*x+y*y+z*z);
	}

	public final static float length( Vector3D A) {
```

```
        return (float) Math.sqrt(A.x*A.x+A.y*A.y+A.z*A.z);
    }

    public final Vector3D normalize() {
        float t=x*x+y*y+z*z;
        if (t!=0 && t!=1) t = (float)(1/Math.sqrt(t));
        return new Vector3D(x*t, y*t, z*t);
    }

    public final static Vector3D normalize( Vector3D A) {
        float t = A.x*A.x+A.y*A.y+A.z*A.z;
        if (t!=0 && t!=1) t =  (float)(1/Math.sqrt(t));

        return new Vector3D(A.x*t, A.y*t, A.z*t);
    }

    public final Vector3D rotate( Vector3D Axis, float angle) {
        Matrix3x3       R=Matrix3x3.rotation(Axis, angle);
        return R.times(this);
    }

    public final Vector3D scale(float s) {
        return new Vector3D(s*x, s*y, s*z);
    }

    public final Vector3D plus( Vector3D A) {
        return new Vector3D(x+A.x, y+A.y, z+A.z);
    }

    public final Vector3D minus( Vector3D A) {
        return new Vector3D(x-A.x, y-A.y, z-A.z);
    }

    public String toString() {
        return new String("["+x+", "+y+", "+z+"]");
    }
}
```

```
import java.awt.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import PlanarReference;
import PlanarView;
import Matrix3x3;
import Vector3D;

public class Vwarp extends Applet implements Runnable
{
        final static float DEFAULT_FOV = 40;
        final static float DEFAULT_MOVE = 1;

        int curRef;                     //index of current reference image
        int numRefs;        //number of reference images available
        PlanarReference reference[];            //array of reference images
        Vector3D eyePoint;                                      //desired center-of-projection
        Vector3D lookAt;                                        //optical axis
        Vector3D up;                                            //three space vector that projects up
        float hfov;                                     //horizontal field-of-view
        float moveScale;                //size of step
        PlanarView          currentView;        //Desired view point and projection
        Raster    window;                       //display window
        Image output;                           //working image

        //the init method is the first one executed when the applet is loaded
        public void init()
        {
                // get the number of refrence images
                try {
                        numRefs = Integer.parseInt(getParameter("numrefs"));
                }catch (NumberFormatException e) {
                        numRefs = 0;
                }

                //read in the reference images
                reference = new PlanarReference[numRefs];
                for (int i=0; i<numRefs; i++) {
                        String refInfo = getParameter("reference"+i);
                        if (refInfo == null) continue;
                        StringTokenizer parser = new StringTokenizer(refInfo, " ,");
                        //get viewing parameters
                        try {
                                Image image = getImage(getDocumentBase(),
parser.nextToken());
                                Image disparity = getImage(getDocumentBase(),
parser.nextToken());
                                Vector3D O = new Vector3D(parser);              //centre-of-
projection
                                Vector3D Vp = new Vector3D(parser);             //vector to image
origin
                                Vector3D Du = new Vector3D(parser);             //x spanning
vector
                                Vector3D Dv = new Vector3D(parser); //y spanning vector
                                float mind, maxd;
```

```
                        mind = Float.valueOf(parser.nextToken()).floatValue();
                        maxd = Float.valueOf(parser.nextToken()).floatValue();
                        Matrix3x3 P = new Matrix3x3(Du, Dv, Vp);
                        PlanarView view = new PlanarView(O, P);

                        reference[i] = new PlanarReference(this, image, disparity,

 maxd, mind, view);
                } catch (NumberFormatException e) {
                        System.err.println("Error: Invalid float value");
                        System.exit(1);
                }
                showStatus("Image["+i+"] size = " + reference[0].width +
                                " x " + reference[0].height);
        }

        //set default view
        curRef = 0;
        eyePoint = reference[curRef].getOrigin();
        lookAt = reference[curRef].getCenter();
        up = new Vector3D(0, 0, 1);
        hfov = DEFAULT_FOV;
        String fovval = getParameter("fov");
        if (fovval != null) {
                try {
                        hfov = Float.valueOf(fovval).floatValue();
                } catch (NumberFormatException e) {
                        hfov = DEFAULT_FOV;
                }
        }
        hfov *= Math.PI/180;

        moveScale = DEFAULT_MOVE;
        String moveval = getParameter("move");
        if (moveval != null) {
                try {
                        moveScale = Float.v alueOf(moveval).floatValue();
                } catch (NumberFormatException e) {
                        moveScale = DEFAULT_MOVE;
                }
        }

        window = new Raster(size().width, size().height);
        window.fill(Color.blue);
        output = window.toImage(this);

        currentView = new PlanarView(eyePoint, lookAt, up, hfov, size());
        warpInfo(ENQUEUE, currentView);
}

//the following variables permits the warper and
// user interface to operate asynchroously
protected final static int ENQUEUE = 0;
protected final static int DEQUEUE = 1;
protected int warpCount = 0;
protected boolean warpOutstanding = false;
```

```
protected PlanarView warpView, nextView;
Thread warper;

protected synchronized boolean warpInfo(int action, PlanarView view) {
        boolean rval = true;
        switch (action) {
        case ENQUEUE:
                nextView = new PlanarView(view);
                warpOutstanding = true;
                warpCount += 1;
                break;
        case DEQUEUE:
                if (warpOutstanding) {
                        warpView = new PlanarView(nextView);
                        warpOutstanding = false;
                } else
                        rval = false;
                break;
        }
        return rval;
}

public void start() {
        warper = new Thread(this);
        warper.start();
}

public void stop() {
        warper.stop();
}

public void run() {
        long time = 0;
        while(true) {
                if (warpInfo(DEQUEUE, null)) {
                        window.fill(getBackground());
                        time = System.currentTimeMillis();
                        reference[curRef].Warp(warpView, window);
                        time = System.currentTimeMillis() - time;
                        output = window.toImage(this);
                        paint(this.getGraphics());
                } else {
                        showStatus(time+"ms Waiting for input["+wa rpCount+"]");
                        try {
                                System.gc();
                                Thread.sleep(100);
                        } catch (InterruptedException e) {
                                break;
                        }
                }
        }
}

public void paint(Graphics g) {
        g.drawImage(output, 0, 0, this);
}
```

```
//simplyfy redraw
public void update(Graphic s g) {
        g.setColor(getForeground());
        paint(g);
}

Vector3D V0;      //vector in the direction
public boolean mouseDown(Event e, int x, int y) {
        V0 = Vector3D.normalize(currentView.map(x,y));
        return true;
}

//handle the drag of the mouse while a b utton is held down
public boolean mouseDrag(Event e, int x, int y) {
        Vector3D V1 = Vector3D.normalize(currentView.map(x,y));
        Vector3D Axis = V1.cross(V0);
        Vector3D newLook = new Vector3D(lookAt);

        //rotate the current look-at direction
        if (Axis.length() > 0.0001) {
                float angle = (float) Math.asin(Vector3D.length(Axis));
                newLook = newLook.rotate(Axis, angle);
        }

        //translate along the look-at direction if either
        //shift or control is pressed during the drag
        if (e.shiftDown()) {
                eyePoint = eyePoint.plus(V0.scale(moveScale));
        } else
        if (e.controlDown()) {
                eyePoint = eyePoint.plus(V0.scale(-moveScale));
        }

        //update the current viewing matrix
        PlanarView view = new PlanarView(eyePoint, newLook, up, hfov, size());
        warpInfo(ENQUEUE, view);
        return true;
}

//handle release of mouse button
public boolean mouseUp(Event e, int x, int y) {
        Vector3D V1 = Vector3D.normalize(currentView.map(x, y));
        Vector3D Axis = V1.cross(V0);

        //rotate the look-at vector
        if (Axis.length() > 0.0001) {
                float angle = (float) Math.asin(Vector3D.length(Axis));
                lookAt = lookAt.rotate(Axis, angle);
        }

        //translate along the look-at direction if either
        //shift or control is pressed during the drag
        if (e.shiftDown()) {
                eyePoint = eyePoint.plus(V0.scale(moveScale));
        } else
```

```
                if (e.controlDown()) {
                        eyePoint = eyePoint.plus(V0.scale(-moveScale));
                }

                //update the current viewing matrix
                currentView = new PlanarView(eyePoint, lookAt, up, hfov, size());
                warpInfo(ENQUEUE,  currentView);
                return true;
        }

        //handle keyboard events
        public boolean keyDown(Event e, int key) {
                switch(key) {
                case 'n':
                        curRef += 1;
                        if (curRef == numRefs) curRef = 0;
                        warpInfo(ENQUEUE, currentView);
                        break;
                case 'r':
                        eyePoint = reference[curRef].getOrigin();
                        lookAt = reference[curRef].getCenter();
                        up = new Vector3D(0,0,1);
                        currentView = new PlanarView(eyePoint, lookAt, up, hfov, size());
                        warpInfo(ENQUEUE, currentView);
                        break;
                case 'w':
                        hfov *= 1.125f;
                        currentView = new PlanarView(eyePoint, lookAt, up, hfov, size());
                        warpInfo(ENQUEUE, currentView);
                        break;
                case 'y':
                        int width=reference[curRef].width;
                        int height=reference[curRef].height;
/*
                        V0 = Vector3D.normalize(eyePoint.minus(lookAt));
                        Vector3D V1 = currentView.map(width,height/2);
                        V1 = Vector3D.normalize(V1.minus(lookAt));
                        Vector3D Axis = V0.cross(V1);
                        float angle = (float) (10*Math.PI/180);
                        eyePoint = eyePoint.rotate(Axis, angle);
*/
                        eyePoint = lookAt.plus(lookAt).minus(eyePoint);
                        currentView = new PlanarView(eyePoint, lookAt, up, hfov, size());
                        warpInfo(ENQUEUE, currentView);
                        break;
                case 'z':
                        hfov *= 0.88888888f;
                        currentView = new PlanarView(eyePoint, lookAt, up, hfov, size());
                        warpInfo(ENQUEUE, currentView);
                        break;
                }

                return true;
        }
}
```